51CTO.com
技术博客Blog

博客月刊

blog.51cto.com

2014年11月
总第 08 期

- 系统管理员常用的Powershell命令
- Linux下高效数据恢复软件extundelete应用实战
- 在Puppet中用ERB模板来自动配置Nginx虚拟主机
- 如何使用swingbench进行oracle数据库压力测试
- 多年以来，你可找到努力的动力？

目录

初学者应该如何学习乃至玩好 Linux 系统呢？

> 作者：余洪春 来源：http://yuhongchun.blog.51cto.com/1604432/1572162

我把之前的一些学习经验和方法跟大家分享下，希望对大家有所帮助：

一、玩好 Linux 一定要经常折腾，说白了，就是动手能力一定要强。我初学 Linux 那块，家里 3 台电脑，我在上面经常反反复复的做 kickstart、网络 ghost、双系统安装的实验。有很长一段时间，我还在其中的一台老式笔记本上安装了 Ubuntu 系统 ,通过它来游览网页和看视频 ,解决各种驱动问题 ,通过这些折腾，对 Linux 也是越来越有兴趣，学习的劲头也越来越足了。

二、床边经常放几本书，临睡觉前或无聊时经常翻一翻，我个人的感觉是夜深人静的时候印象非常深刻，很多知识点很容易就记住了。

三、我习惯手边放一个小本，初学的一些 Linux 操作单词我会写在上面，详细用法也会记载，等人或吃饭的时候我会拿来翻一翻，这样感觉掌握得特别快。对英文头疼的同学建议坚持看中英文字幕的美剧，比如现在流行的《生活大爆炸》、《傲骨贤妻》、《权力的游戏》等等，相信英文不会成为学习的阻碍了。

四、实验过程中的排障一定要注意出错的原因，比如我近期发现自己 PXE 安装的实验机器，老是带了一个 ifcfg_eth0.bak 文件，后来经过仔细分析，发现是由于我的机器是 Kickstart 安装，分配的 MAC 跟原来机器不一致，机器重启 service 服务以后就自动的添加了一个 ifcfg_eth0.bak 文件，知道故障的原因以后就好办了。工作中遇到的问题，也应该反反复复排查，千万不要在没搞清出错原因的前提下胡乱猜测，这样的效果是非常糟糕的。大家可以看下有问题的网卡文件，下面分配的 MAC 地址实际跟系统网卡自身的 MAC 地址并不是相匹配的，如下所示：

```
[root@localhost network-scripts]# cat ifcfg-eth0.bak
# Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Express Gigabit Ethernet controller
DEVICE=eth0
BOOTPROTO=none
HWADDR=fe:ff:ff:ff:ff:ff
ONBOOT=yes
NETMASK=255.255.255.0
IPADDR=192.168.1.120
GATEWAY=192.168.1.1
```

```
TYPE=Ethernet
```

而实际的网卡 MAC 地址我们用 ifconfig eth0 可以查看得到 ,这个跟上面所列的网卡 MAC 确实是不一样的，如下所示：

```
[root@localhost ~]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 90:2B:34:87:F3:CD
```

五、如果遇到复杂的问题或是自己想了很久也没有答案的知识点，建议可以去看一下别人的博客，学习别人的实验和心得体会，再融会贯通，吸收了就成了自己的。现在技术论坛的活跃度不高，但很人个人技术含金量还是很高的。这里建议大家一定要做好相关的知识难点的笔记，好记性不如烂笔头，一个一个小知识，长期坚持下去就是一个很可观的数值了。

六、**实践出真知，在阅读别人的技术文章或著作时，我也发现了不少错误之处，这时候千万不要相信所谓的权威**（笔者手上正在阅读的一本国外专家著作中就存在着不少问题），相信自己的实验结果，一切以其为判断依据。

七、**遇到新技术或疑难问题，先实验，再原理**，不明白这点的同学先按照我的这种方法试一试，慢慢就明白了。

这些方法贵在坚持，持之以恒的话，肯定是有收获的。

系统管理员常用的 Powershell 命令

作者：beanxyz 来源：http://beanxyz.blog.51cto.com/5570417/1571507

上周豆子参加了微软 Teched 2014 Sydney 的 2 天会议。这个会议包括了 50 多个讲座 ,包括了开发，架构，移动 3 个大的方向。其中一个 300 级别的讲座介绍了一些系统管理常用的命令。这些命令，豆子绝大部分都很熟悉，这里再温故知新一下。作为一个系统管理员，一般都对长篇大论的脚本，各种参数，循环，判断语句，输入输出等等是敬而远之的~但是短小精悍的命令行语句还是可以试试的。

开始之前，先说两个最最基本的命令。Get-Command 和 help。

Get-command 可以搜索相关命令，help 可以搜索具体的例子。

比如我想创建新的虚拟机，但是我不知道有哪些命令，那么搜索一下好了

```
PS C:\WINDOWS\system32> get-command "*vm"

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Checkpoint-VM                                      Hyper-V
Cmdlet          Compare-VM                                         Hyper-V
Cmdlet          Debug-VM                                           Hyper-V
Cmdlet          Export-VM                                          Hyper-V
Cmdlet          Get-VM                                             Hyper-V
Cmdlet          Import-VM                                          Hyper-V
Cmdlet          Measure-VM                                         Hyper-V
Cmdlet          Move-VM                                            Hyper-V
Cmdlet          New-VM                                             Hyper-V
Cmdlet          Remove-VM                                          Hyper-V
Cmdlet          Rename-VM                                          Hyper-V
Cmdlet          Repair-VM                                          Hyper-V
Cmdlet          Restart-VM                                         Hyper-V
Cmdlet          Resume-VM                                          Hyper-V
Cmdlet          Save-VM                                            Hyper-V
Cmdlet          Set-VM                                             Hyper-V
Cmdlet          Start-VM                                           Hyper-V
Cmdlet          Stop-VM                                            Hyper-V
Cmdlet          Suspend-VM                                         Hyper-V
```

搜索了一堆命令出来，那么怎么用呢，查看一下帮助文档，如果不想看具体的语法，直接输入 -examples 看看例子好了。然后就可以直接复制粘贴来使用了。

```
PS C:\WINDOWS\system32> help new-vm -examples

NAME
    New-VM

SYNOPSIS
    Creates a new virtual machine.

    Example 1

    PS C:\> New-VM -Name "new 1" -MemoryStartupBytes 512MB


    This example creates a new virtual machine named new 1 that has 512 MB of memory.
    Example 2

    PS C:\> New-VM -Name "new 2" -MemoryStartupBytes 1GB -NewVHDPath d:\vhd\base.vhdx


    This example creates a virtual machine named new 2 that has 1 GB of memory and that is connected to a new 40 GB
    virtual hard disk that uses the VHDX format.
    Example 3

    PS C:\> New-VM -Name "new 3" -MemoryStartupBytes 1GB -VHDPath d:\vhd\BaseImage.vhdx


    This example creates a virtual machine named new 3 that has 1 GB of memory and connects it to an existing virtual
    hard disk that uses the VHDX format.
```

同时呢，工欲善其事，必先利其器。帮助文档需要更新到最新版本。

更新帮助文档很简单，输入 help-update，就会更新了。

```
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

Updating Help for module Defender
    Downloading Help Content...
    [ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

知道了这两个最基本的命令，我们再来看看正题，系统管理常见的 30 多个操作都可以用 powershell 实现。

网络基本配置和查看：

1. 系统管理员用来查询 IP 地址一般都习惯用 ipconfig /all，现在可以用 Get-NetIPConfiguration -detailed 来替代了。

```
PS C:\Users\yuan.li.AP-MEDIA> Get-NetIPConfiguration -Detailed


ComputerName                : CG11D2S
InterfaceAlias              : vEthernet (External Virtual Switch)
InterfaceIndex              : 7
InterfaceDescription        : Hyper-V Virtual Ethernet Adapter #2
NetAdapter.LinkLayerAddress : 18-03-73-1B-50-4B
NetAdapter.Status           : Up
NetProfile.Name             : global.loc
NetProfile.NetworkCategory  : DomainAuthenticated
NetProfile.IPv6Connectivity : LocalNetwork
NetProfile.IPv4Connectivity : Internet
IPv6LinkLocalAddress        : fe80::7da2:a07e:a392:d6dd%7
IPv4Address                 : 10.71.82.249
IPv6DefaultGateway          :
IPv4DefaultGateway          : 10.71.82.254
NetIPv6Interface.NlMTU      : 1500
NetIPv4Interface.NlMTU      : 1500
NetIPv6Interface.DHCP       : Enabled
NetIPv4Interface.DHCP       : Enabled
DNSServer                   : 10.71.65.19
                              10.71.65.20
```

2. 查看网卡信息

Get-NetAdapter

Get-NetAdapterStatistics

Get-NetIPAddress

```
PS C:\Users\yuan.li.AP-MEDIA> Get-NetAdapter

Name                      InterfaceDescription                    ifIndex Statu
                                                                          s
----                      --------------------                    ------- -----
vEthernet (External Vi... Hyper-V Virtual Ethernet Adapter #2           7 Up
Ethernet                  Intel(R) 82579LM Gigabit Network Con...       3 Up

PS C:\Users\yuan.li.AP-MEDIA> Get-NetAdapterStatistics

Name                            ReceivedBytes ReceivedUnicastPackets SentBytes
----                            ------------- ---------------------- ---------
vEthernet (External Virtual...    12277363944               16172872 ...470305
```

3. 配置 IP 地址 New-NetIPAddress

这个命令一般是 Windows 2012 core server 下初始化配置的时候可以使用，当然也可以用 sconfig 或者传统的 netsh interface 命令

```
PS C:\Users\yuan.li.AP-MEDIA> New-NetIPAddress -InterfaceAlias Ethernet -IPAddre
ss 10.71.82.249 -PrefixLength 24 -DefaultGateway 10.71.82.254
```

这个命令看起来比较啰嗦，参数比较多，就像上面说的，如果记不清，那么看看帮助的例子就行了。

```
PS C:\WINDOWS\system32> get-command "*ipaddress"

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Function        Get-DhcpServerv4FreeIPAddress                      DhcpServer
Function        Get-DhcpServerv6FreeIPAddress                      DhcpServer
Function        Get-NetIPAddress                                   NetTCPIP
Function        New-NetIPAddress                                   NetTCPIP
Function        Remove-NetIPAddress                                NetTCPIP
Function        Set-NetIPAddress                                   NetTCPIP


PS C:\WINDOWS\system32> help New-NetIPAddress -examples

NAME
    New-NetIPAddress

SYNOPSIS
    Creates and configures an IP address.

    Example 1: Add an IPv4 address

    PS C:\>New-NetIPAddress -InterfaceIndex 12 -IPAddress 192.168.0.1 -PrefixLength 24 -DefaultGateway 192.168.0.5


    The second command removes the IPv4 address. To remove the IPv4 address, use the Remove-NetIPAddress cmdlet.
    PS C:\>Remove-NetIPAddress -IPAddress 192.168.0.1 -DefaultGateway 192.168.0.5


    The first command adds a new IPv4 address to the network interface at index 12. The PrefixLength parameter
    specifies the subnet mask for the IP address. In this example, the PrefixLength of 24 equals a subnet mask of
    255.255.255.0.  When you add an IPv4 address, the address specified for the Default Gateway must be in the same
    subnet as the IPv4 address that you add.
```

4. 配置 DNS

Set-DNSClientServerAddress

类似的，这个在 server core 下面也是常见的命令，当然省事也可以通过 sconfig 配置,或者 DOS 命令 netsh interface 实现。

```
PS C:\WINDOWS\system32> help Set-DnsClientServerAddress -examples

NAME
    Set-DnsClientServerAddress

SYNOPSIS
    Sets DNS server addresses associated with the TCP/IP properties on an interface.

    EXAMPLE 1

    PS C:\>Set-DnsClientServerAddress -InterfaceIndex 12 -ServerAddresses ("10.0.0.1","10.0.0.2")


    This example sets the DNS server addresses on a specified interface with the index value of 12.
    EXAMPLE 2

    PS C:\>Set-DnsClientServerAddress -InterfaceIndex 12 -ResetServerAddresses


    This example resets the DNS client to use the default DNS server addresses specified by DHCP on the interface with
    an index value of 12.
```

网络查错

传统的排错第一步都是通过 PING，Tracert 或者 Telnet 判断路由和端口是否打开。比如

ping 10.1.1.1

tracert www.baidu.com

telnet 8.8.8.8 53

Powershell 下面一个命令都实现了。

5. PING 远程服务器

Test-NetConnection www.google.com

```
PS C:\WINDOWS\system32> Test-NetConnection www.baidu.com


ComputerName            : www.baidu.com
RemoteAddress           : 180.76.3.151
InterfaceAlias          : vEthernet (External Virtual Switch)
SourceAddress           : 10.71.82.249
PingSucceeded           : True
PingReplyDetails (RTT) : 115 ms
```

6. Telnet 端口

```
PS C:\WINDOWS\system32> Test-NetConnection 8.8.8.8 -Port 53


ComputerName            : 8.8.8.8
RemoteAddress           : 8.8.8.8
RemotePort              : 53
InterfaceAlias          : vEthernet (External Virtual Switch)
SourceAddress           : 10.71.82.249
PingSucceeded           : True
PingReplyDetails (RTT) : 2 ms
TcpTestSucceeded        : True
```

7. 跟踪路径

```
PS C:\WINDOWS\system32> Test-NetConnection 8.8.8.8 -TraceRoute


ComputerName            : 8.8.8.8
RemoteAddress           : 8.8.8.8
InterfaceAlias          : vEthernet (External Virtual Switch)
SourceAddress           : 10.71.82.249
PingSucceeded           : True
PingReplyDetails (RTT) : 2 ms
TraceRoute              : 10.71.82.254
                          TimedOut
                          TimedOut
                          TimedOut
                          TimedOut
                          TimedOut
                          8.8.8.8
```

Windows 服务的操作

这几个操作很简单直接

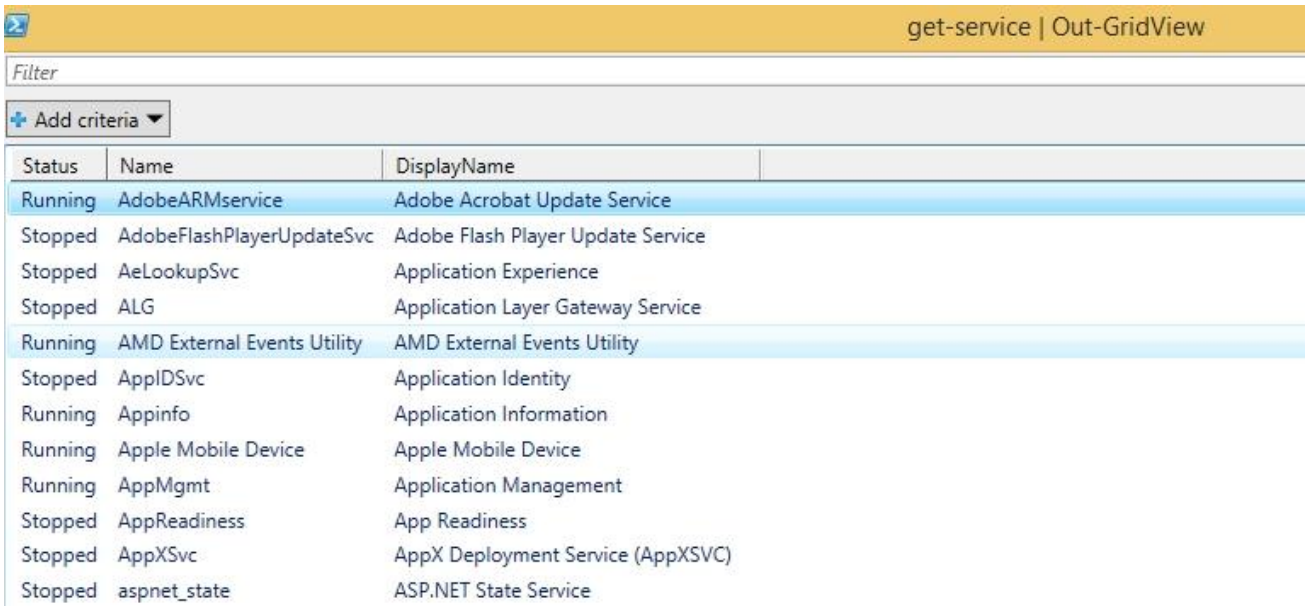
8. 重启服务

restart-service

9.获取当前的服务

get-service| out-gridview

注意那个 out-gridview 的输出格式，所有的 PS 命令都是通用的，可以把结果用表格的性格输出来，这样用户还可以手动的排序或者添加标准（criteria），是不是很简单



10.停止，开始，配置服务

stop-service

start-service

set-service

AD 和域的操作，这个可能是 windows 2012 系统管理员最应该记住的部分吧

11. 计算机的重命名，如果通过图形界面操作，需要右击电脑，然后属性，然后 高级属性设置，然后计算机名里面从能修改，通过命令行直接就修改了，这个在 server core 下面也是初始化设置必须的，更改名字，重启，然后加入域

Rename-Computer XXX

12. 重启电脑 restart-computer，当然传统的 shutdown /r /f /t 我也觉得蛮好的

13. 关机 shutdown-computer

14. 加入域 Add-computer -domainname test.com

15. 修复 AD 的信任关系。这个命令对豆子来说都是很新的命令。传统的如果某台计算机无法验证 AD

最简单的解决方法是退出域，重启，然后重新加入域，重启。现在可以在计算机上直接执行以下命令进行修复。

如果直接执行，他会进行判断，True 表示 OK； False 表示无法连接 AD，那么需要提供管理员密码进行修复

例如

test-computersecurechannel -credential domain\admin -repair

```
PS C:\WINDOWS\system32> Test-ComputerSecureChannel
True
PS C:\WINDOWS\system32> help Test-ComputerSecureChannel

NAME
    Test-ComputerSecureChannel

SYNOPSIS
    Tests and repairs the secure channel between the local computer and its domain.


SYNTAX
    Test-ComputerSecureChannel [-Credential <PSCredential>] [-Repair] [-Server <String>] [-Confirm] [-WhatIf]
    [<CommonParameters>]


DESCRIPTION
    The Test-ComputerSecureChannel cmdlet verifies that the secure channel between the local computer and its domain
    is working correctly by checking the status of its trust relationships. If a connection fails, you can use the
    Repair parameter to try to restore it.

    Test-ComputerSecureChannel returns "True" if the secure channel is working correctly and "False" if it is not.
    This result lets you use the cmdlet in conditional statements in functions and scripts. To get more detailed test
    results, use the Verbose parameter.

    This cmdlet works much like NetDom.exe. Both NetDom and Test-ComputerSecureChannel use the NetLogon service to
    perform the actions.
```

16. 配置防火墙 profile

set-netfirewallprofile

比如最简单的例子，打开 domain，public 和 private 的防火墙

```
    Set-NetFirewallProfile

SYNOPSIS
    Configures settings that apply to the per-profile configurations of the Windows Firewall with Advanced Security.

    Example 1

    PS C:\>Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled True
```

17，配置防火墙策略,很长很长的命令，记不着的话就看看帮助 help -examples

New-netfirewallrule，可以设置出去和进来的请求服务

比如第一个禁止所有出去的 80 端口，换句话说，上不了网了

第二个禁止所有来自 Wins 服务器的请求

```
NAME
    New-NetFirewallRule

SYNOPSIS
    Creates a new inbound or outbound firewall rule and adds the rule to the target computer.

    EXAMPLE 1

    PS C:\> New-NetFirewallRule -DisplayName "Block Outbound Port 80" -Direction Outbound -LocalPort 80 -Protocol TCP
    -Action Block


    This example creates an outbound firewall rule to block all of the traffic from the local computer that originates
    on TCP port 80.
    EXAMPLE 2

    PS C:\> New-NetFirewallRule -DisplayName "Block WINS" -Direction Inbound -Action Block -RemoteAddress WINS


    This example creates a firewall rule that blocks all inbound traffic from all WINS servers.
```

18. 添加 Roles 和 Features

Install-windowsfeature

比如说 ,windows 2012 下面默认安装.Net 3.5 是安装不了的 ,必须指定对应的路径, 当然也可以通过 GPO

指定默认安装路径，然后把对应的文件事先放在那里也是不错的解决方案。

Install-windowFeature net-framework-core -source d:\sources\sxs

19.重置 AD 用户的密码

这几个命令很方便，保存下来，就不用进 AD Users and Groups 里面去修改密码了。

首先可以利用 Converto-securestring 设置一个加密的密码

```
PS C:\Users\yuan.li#> $newpwd=ConvertTo-SecureString "Horse201410$" -AsPlainText -force
```

然后通过 set-ADAccountPassword 重置 密码。 最后那个 passthru 的目的是显示账号信息

```
PS C:\Users\yuan.li#> Set-ADAccountPassword yuan.li -NewPassword $newpwd -reset -PassThru


DistinguishedName : CN=Yuan Li,OU=Sydney,OU=Users,OU=DentsuAegis,OU=Companies,OU=ANZ,DC=ap,DC=media,DC=global,DC=loc
Enabled           : True
Name              : Yuan Li
ObjectClass       : user
ObjectGUID        : f787b579-01f7-44e6-b906-48fd70c6acec
SamAccountName    : yuan.li
SID               : S-1-5-21-3646886372-3364824601-2182576161-156351
UserPrincipalName : yuan.li@carat.com
```

还可以更进一步，要求用户下次登录的时候必须修改密码

```
PS C:\Users\yuan.li#> Set-ADAccountPassword yuan.li -NewPassword $newpwd -reset -PassThru | Set-ADUser -ChangePasswordAt
Logon $true
```

关于 AD 的操作 ,肯定绕不过 FSMO 的配置。几乎所有的 Windows 系统管理面试都会涉及这个问题。

传统的配置可以通过 GUI，也可以通过 ntsutil 命令来配置。Powershell 提供了更简单的方式。

豆子曾经写过一篇博客具体比较这几种方式 http://beanxyz.blog.51cto.com/5570417/1313693

这里就不赘述了

20. 定位 FSMO

基本格式如下

Get-ADForest test.com | FT SchemaMaster

Get-ADForest test.com | FT RidMaster

21. 迁移 FSMO

Move-ADDirectoryserverOperationMasterRole

配置服务器，还需要打开远程桌面。

22. 打开 RDP 桌面，这个其实有好几个地方可以配置。 最直观的是通过 sconfig，也可以修改注册表，

或者打开防火墙策略(3389 端口）

set-itemProperty -path 'hklm:\system\currentcontrolset\control\terminal server' =name

"fdenytsconnections" -value 0

enable-netfirewallrule -displaygroup "remote desktop"

```
PS C:\WINDOWS\system32> Enable-NetFirewallRule -DisplayGroup "Remote Desktop" -passthru

Name                  : RemoteDesktop-UserMode-In-UDP
DisplayName           : Remote Desktop - User Mode (UDP-In)
Description           : Inbound rule for the Remote Desktop service to allow RDP traffic. [UDP 3389]
DisplayGroup          : Remote Desktop
Group                 : @FirewallAPI.dll,-28752
Enabled               : False
Profile               : Public
Platform              : {}
Direction             : Inbound
Action                : Allow
EdgeTraversalPolicy   : Block
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                 :
PrimaryStatus         : OK
Status                : The rule was parsed successfully from the store. (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType : Local

Name                  : RemoteDesktop-Shadow-In-TCP
DisplayName           : Remote Desktop - Shadow (TCP-In)
Description           : Inbound rule for the Remote Desktop service to allow shadowing of an existing Remote Desktop
                        session. (TCP-In)
DisplayGroup          : Remote Desktop
Group                 : @FirewallAPI.dll,-28752
Enabled               : False
Profile               : Public
Platform              : {}
Direction             : Inbound
Action                : Allow
EdgeTraversalPolicy   : DeferToApp
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                 :
PrimaryStatus         : OK
Status                : The rule was parsed successfully from the store. (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType : Local

Name                  : {96F34309-7C55-44F2-B6B4-6F82B3DA69CD}
DisplayName           : Remote Desktop - User Mode (TCP-In)
Description           : Inbound rule for the Remote Desktop service to allow RDP traffic. [TCP 3389]
```

23. 查看 hotfix

一般企业里面都是通过 WSUS 来推送 Hotfix，不过有的时候计算机并不是 100%可以成功获取的。可以通过 get-hotfix 来进行判断

```
PS C:\WINDOWS\system32> get-hotfix

Source        Description      HotFixID      InstalledBy          InstalledOn
------        -----------      --------      -----------          -----------
CG11D2S       Update           KB2899189_... NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2693643     MITCH\yuan.li_old    2/07/2014 12:00:00 AM
CG11D2S       Update           KB2843630     NT AUTHORITY\SYSTEM
CG11D2S       Security Update  KB2868626     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2883200     CG11D2S\anz-admin...
CG11D2S       Update           KB2887595     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2889543     CG11D2S\anz-admin...
CG11D2S       Security Update  KB2893294     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2894029     CG11D2S\anz-admin...
CG11D2S       Update           KB2894179     CG11D2S\anz-admin...
CG11D2S       Security Update  KB2894852     NT AUTHORITY\SYSTEM  10/06/2014 12:00:00 AM
CG11D2S       Security Update  KB2894856     NT AUTHORITY\SYSTEM  10/06/2014 12:00:00 AM
CG11D2S       Security Update  KB2900986     NT AUTHORITY\SYSTEM
CG11D2S       Security Update  KB2901128     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2903939     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2911106     NT AUTHORITY\SYSTEM
CG11D2S       Security Update  KB2912390     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2913152     NT AUTHORITY\SYSTEM
CG11D2S       Security Update  KB2918614     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2919355     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2919394     NT AUTHORITY\SYSTEM
CG11D2S       Security Update  KB2920189     NT AUTHORITY\SYSTEM
CG11D2S       Update           KB2923528     NT AUTHORITY\SYSTEM
CG11D2S       Security Update  KB2928120     NT AUTHORITY\SYSTEM
```

24. 查看密码永不过期的账号

管理员常常把自个的账号设置为不过期，尽管我们要求用户每隔 60 天就必须 reset 一次。很多服务的运行账号也是这样。怎么搜索这些账号呢？ ADUC 是可以直接搜索的，Powershell 也可以。

```
PS C:\WINDOWS\system32> Search-ADAccount -PasswordNeverExpires | out-gridview
```

25. 类似上面的方法，我们可以搜索最近没有登录的账号，disable 的账号，过期的账号，即将过期账号等等

比如，这个会搜未来 6 天内会过期的账号

```
PS C:\>Search-ADAccount -AccountExpiring -TimeSpan 6.00:00:00 | FT Name,ObjectClass -A
Name           ObjectClass
----           -----------
Iulian Calinov user
John Campbell  user
Garth Fort     user
```

Hyper-V 虚拟机

Hyper-V 是 2012 最重要的新功能，毕竟微软的虚拟化平台和私有云都是以这个为基础的。TechED 里面专门有一个讲座来讨论如何更好的在 Hper-V 下面使用 Powershell。

豆子使用的 windows 8.1 Powershell 下面有 167 个命令。 meaure 可以返回一共多少行记录

```
PS C:\WINDOWS\system32> Get-Command -module Hyper-v "*vm*" | measure


Count    : 167
Average  :
Sum      :
Maximum  :
Minimum  :
Property :
```

这么多指令就不一一阐述了，大概看看几个最基本的命令

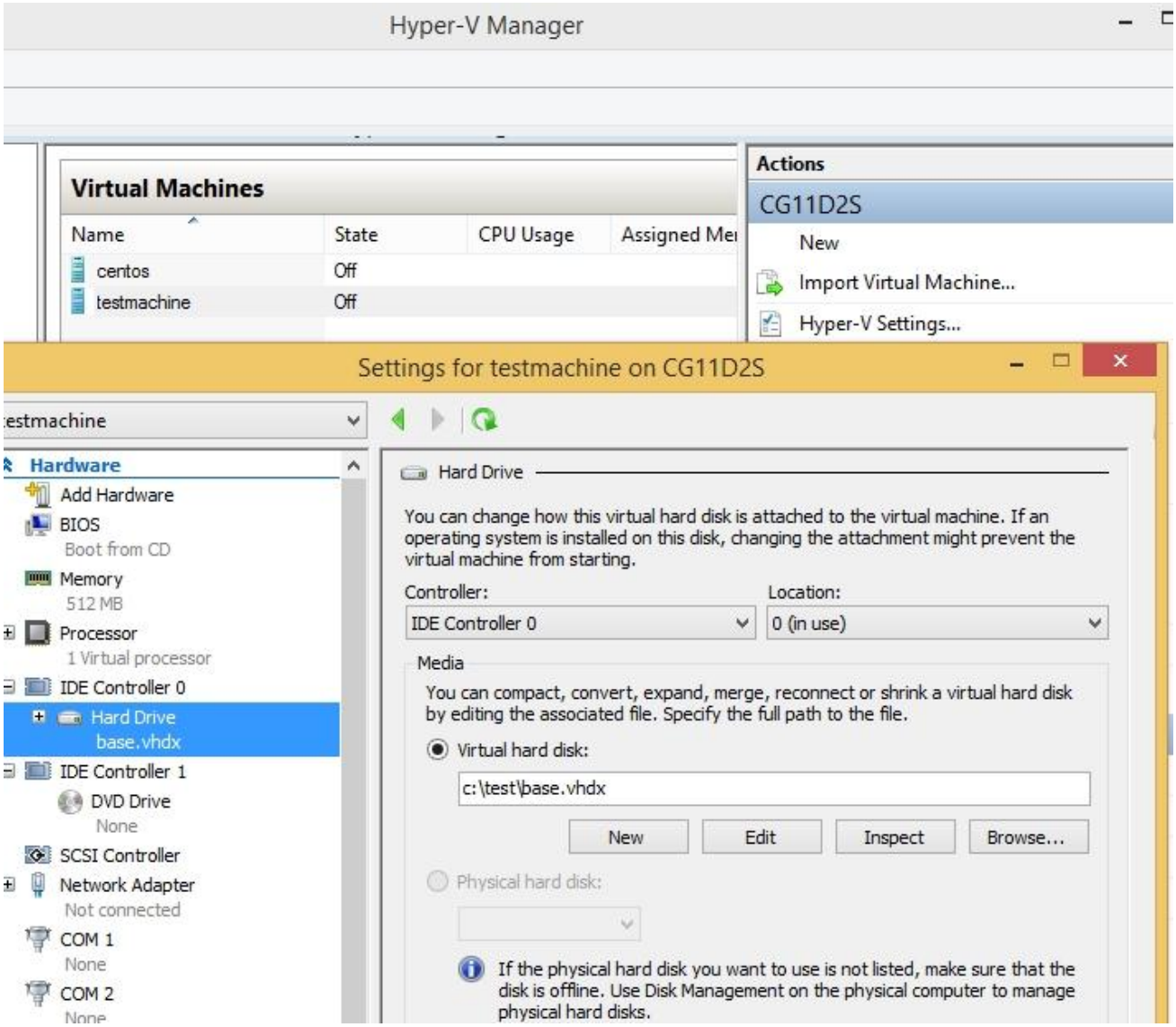
创建虚拟机

26. New-VM

```
PS C:\WINDOWS\system32> new-vm -name "testmachine" -MemoryStartupBytes 512MB -NewVHDPath "c:\test\base.vhdx" -NewVHDSize
Bytes 10GB

Name        State CPUUsage(%) MemoryAssigned(M) Uptime   Status
----        ----- ----------- ----------------- ------   ------
testmachine Off   0           0                 00:00:00 Operating normally
```

打开 Hyper-V Manager，可以看见已经创建了虚拟机



27. 配置虚拟网络

创建虚拟机之后，还需要分配网络。首先看看目前有哪些虚拟机

Get-VM

```
PS C:\WINDOWS\system32> get-vm

Name        State CPUUsage(%) MemoryAssigned(M) Uptime   Status
----        ----- ----------- ----------------- ------   ------
centos      Off   0           0                 00:00:00 Operating normally
testmachine Off   0           0                 00:00:00 Operating normally
```

在这个基础上还可以继续获取虚拟网卡的信息 Get-VMNetworkAdapter

可以看见新创建的虚拟机目前没有分配网络

```
PS C:\WINDOWS\system32> get-vm | Get-VMNetworkAdapter

Name            IsManagementOs VMName      SwitchName              MacAddress   Status IPAddresses
----            -------------- ------      ----------              ----------   ------ -----------
Network Adapter False          centos      External Virtual Switch 00155D52F902        {}
Network Adapter False          testmachine                         000000000000        {}
```

在这个基础上使用 Connect-VMNetworkAdapter 就可以绑定交换机网络了

```
PS C:\WINDOWS\system32> get-vm | Get-VMNetworkAdapter | Connect-VMNetworkAdapter -SwitchName 'External Virtual Switch'
```

再验证一下，已经分配好了

```
PS C:\WINDOWS\system32> get-vm | Get-VMNetworkAdapter

Name            IsManagementOs VMName      SwitchName              MacAddress   Status IPAddresses
----            -------------- ------      ----------              ----------   ------ -----------
Network Adapter False          centos      External Virtual Switch 00155D52F902        {}
Network Adapter False          testmachine External Virtual Switch 000000000000        {}
```

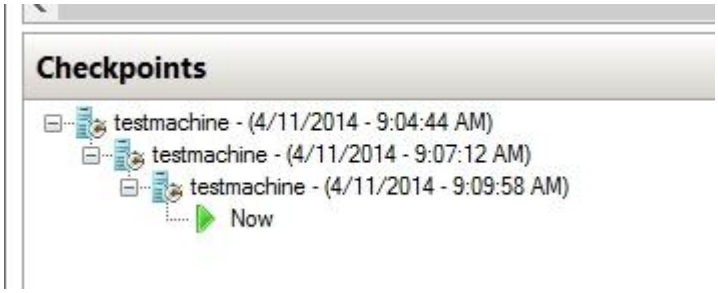
28. 创建还原点

get-VM | checkpoint-VM 就可以对指定的 VM 创建还原点了

```
PS C:\WINDOWS\system32> get-vm testmachine

Name        State CPUUsage(%) MemoryAssigned(M) Uptime   Status
----        ----- ----------- ----------------- ------   ------
testmachine Off   0           0                 00:00:00 Operating

PS C:\WINDOWS\system32> get-vm testmachine | checkpoint-vm
PS C:\WINDOWS\system32>
```



系统性能排查命令及优化思路

> 作者：兰心 hui 性 来源：http://harisxiong.blog.51cto.com/7513022/1569034

最近笔者经常处理了一些线上的问题机器。特抽空写一篇文章将处理系统性能问题和优化思路进行总结，方便后续工作中系统故障的排查。作为运维，收到网管系统性能报警应该是常有的事情。而快速进行问题定位并解决则是工作的关键。我们在排查或者优化一个系统的时候无外乎从以下几个方面考虑：

1.CPU 的使用率异常，如某个核心的使用率过高，而其它核心则处于空闲的状况。

2.内存的使用状况：通常需要注意是否程序内存泄漏导致的。

3.磁盘 IO 性能：通常磁盘 IO 不正常时我们需要判断程序是否处在顺序读写的状态。

4.网络 IO 负载：如 web 服务要考虑 time_wait 状态的连接数；如代理服务器，检查系统打开的 socket 连接数。

5.程序性能问题：缓存和 DB 类注关注写的性能，web 服务类关注内存的使用

6.系统 BUG：一般都是在某些高并发场景才体现出来问题。

常用的判断工具和命令的用法总结如下：

1.系统性能综合判断工具 vmstat

当系统出现性能瓶颈后可以使用 vmstat 命令简单判断一下进程的运行状况及系统资源的使用率，详细输入情况如下：

```
[root@node30 ~]# vmstat 3 10
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 326400  13292  52584    0    0   306    28  141  112  1  4 90  5  0
 0  0      0 326376  13292  52584    0    0     0     0   15   13  0  0 100  0  0
 0  0      0 326376  13292  52584    0    0     0     0   13   15  0  0 100  0  0
 0  0      0 326376  13292  52584    0    0     0     0   13   13  0  0 100  0  0
```

注意事项：

1.r 和 b 的数量，一般 r 小于 CPU 的核心数，b 表示阻塞的队列长度。如果 r 并未达到最大值而 b 的数量较多，我们就需要观察进程所需要的资源状况。

2.注意 in 和 cs 的比例和数量，in 表示中断的请求数，而 cs 才代表 CPU 对中断的处理状况。如果有较多的上下文切换则可以结合 top 命令观察 CPU 性能。

选项解释：

```
r     running 状态进程个数，小于 CPU 核心数（CPU 时间片 + 所需要的资源）
b     被阻塞的进程队列的长度（等待 I/O 完成即进程所需要的资源未到位或者未拿到 CPU 时间片）
swpd  从物理内存交换至交换分区的数据量
free  空闲物理内存空间
buff  buffer  cache 的空间大小，即 IO 处理时的缓冲。（结果）
cache page cache 的空间大小，缓存的是 linux 中的具体文件。进程运行时所需要的资源。（过程）
si    从内存转到 swap 分区
so    从 swap 到内存(kb/s)
bi    从块设备读入内存的数据量(kb/s)，如 select
bo    从内存保存至块设备的数据量(kb/s)，如 update，insert
in    中断发生的速率，通常意味每秒多少次中断请求发生产生原因：时间片轮转，IO（磁盘/网络）
cs    对中断的响应，即上下文切换（进程切换）个数
```

2.LinuxCPU 使用状况排查命令 top(htop)

```
top - 21:00:19 up 218 days,  2:47,  1 user,  load average: 1.21, 0.70, 0.32
Tasks: 170 total,   1 running, 169 sleeping,   0 stopped,   0 zombie
Cpu0  :  1.0%us,  0.7%sy,  0.0%ni, 97.7%id,  0.3%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu1  :  0.7%us,  0.3%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  1.3%us,  0.3%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu3  : 16.3%us,  1.0%sy,  0.0%ni, 82.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32711396k total, 32545628k used,   165768k free,   327244k buffers
Swap:  2101192k total,  1008420k used,  1092772k free,  9033084k cached
```

注意事项：

1.键入 1 观察各 CPU 使用状况，可能会出现某些核心使用率较高而某些核心却处于空闲状态。如果运行类似于 nginx 这样的 web 应用可以将核心与进程绑定，这样既能将用户请求负载均衡也能减小 CPU 上下文切换的消耗。

2.使用该命令一般会查看每个 CPU 的使用率。如果发现 id 资源较少，也可以观察 wa 状态百分比，该状态高百分比也有可能是程序未能拿到运行时所需要的资源或者时间片。所以有的时候我们发现系统 load 较高，而 cpu use 较少的情况。都是因为进程在等待所运行资源的到位。

3.可以键入 M 基于内存使用大小排序观察各进程内存使用大小。

选项解释：

us	sy	ni	id	wa	hi	si	st
用户空间	内核空间	优先级	空闲	等待I/O完成	硬中断占据%	软中断占据%	被偷走时间（硬件虚拟化）
PID	PR	NI	VIRT	RES	SHR	S	TIME+(1/100s)
进程号	优先级	nice	虚拟内存	实际内存	共享内存	状态	进程占用时间总计

3.内存信息展示 free

```
[root@node30 ~]# free -m
             total       used       free     shared    buffers     cached
Mem:           482        341        140          0         49        152
-/+ buffers/cache:        140        342
Swap:         2047          0       2047
```

注意事项：

1.根据 buffers 和 cached 的含义，一般我们认为系统可用内存大小为：free + cached

2.linux 内存处理机制:程序在运行的过程中，进程会加载程序所需要的资源到内存，而当进程运行结束其所 cache 的资源也不会被释放。当其它程序需要资源时，才会根据策略对 cache 中的内存进行清理。这种内存使用机制有利于提高利用共享内存通信程序的效率。

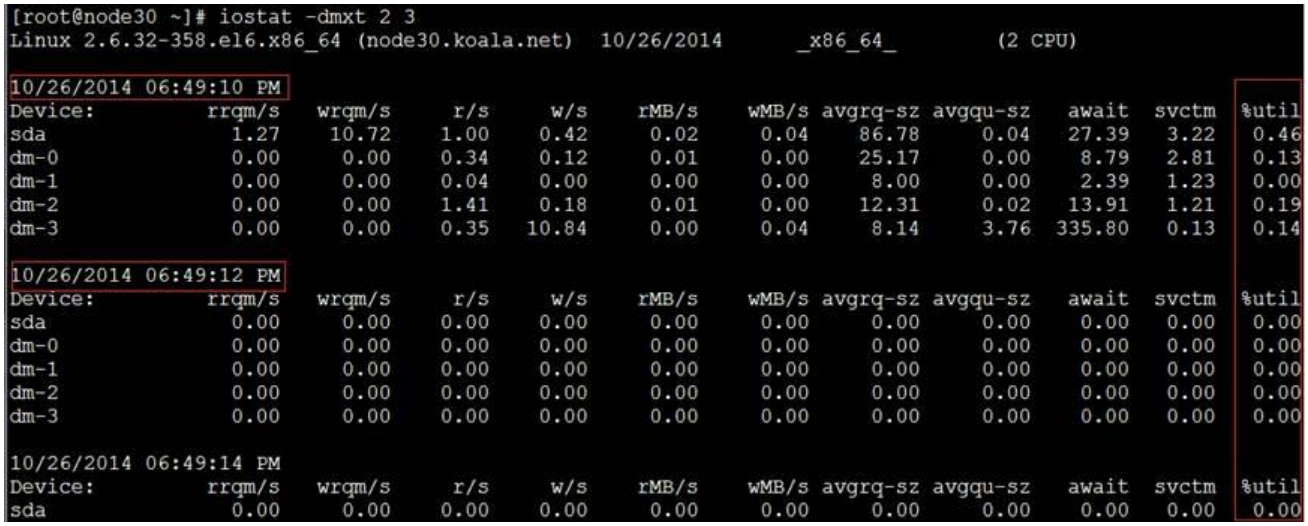
4.实时内核 slab 缓存信息展示

Linux 内核需要为临时对象如任务或者设备结构和节点分配内存 ,缓存分配器管理着这些类型对象的缓存。现代 Linux 内核部署了该缓存分配器以持有缓存 ,称之为片。不同类型的片缓存由片分配器维护。slabtop 命令，该命令显示了实时内核片缓存信息。

```
[root@node30 ~]# cat /proc/meminfo | grep -A 2 Slab
Slab:             100152 kB
SReclaimable:      43128 kB
SUnreclaim:        57024 kB
```

SReclaimable 代表可回收内存的大小，可以使用 slabtop -s c 选项基于 cache size 进行排序展示各组件的 slab 使用状况。

5.系统 IO 判断命令 iostat



```
[root@node30 ~]# iostat -dmxt 2 3
Linux 2.6.32-358.el6.x86_64 (node30.koala.net)  10/26/2014      _x86_64_        (2 CPU)

10/26/2014 06:49:10 PM
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               1.27    10.72    1.00    0.42     0.02     0.04    86.78     0.04   27.39   3.22   0.46
dm-0              0.00     0.00    0.34    0.12     0.01     0.00    25.17     0.00    8.79   2.81   0.13
dm-1              0.00     0.00    0.04    0.00     0.00     0.00     8.00     0.00    2.39   1.23   0.00
dm-2              0.00     0.00    1.41    0.18     0.01     0.00    12.31     0.02   13.91   1.21   0.19
dm-3              0.00     0.00    0.35   10.84     0.00     0.04     8.14     3.76  335.80   0.13   0.14

10/26/2014 06:49:12 PM
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
dm-2              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
dm-3              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00

10/26/2014 06:49:14 PM
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
```

注意事项：

1.类似 iostat 这种交互式命令，我们通常都会观察取得第二次往后的数据。上图中最后一列表示 1s 中有百分之多少的时间用于 I/O 操作。或者说 1s 中有多少时间 IO 队列是非空的。

2.如果%util 接近 100%，说明产生的 IO 请求太多，IO 系统已经满负荷，该磁盘可能存在性能瓶颈。可考虑程序是否为顺序读写。

3.idle 小于 70%时，如果 wa 较多，伴随 IO 就会有较大的压力，可以结合 vmstat 查看 b 参数（等待资源的进程数）和 wa 参数（资源及时间片）排查。

选项解释：

```
          rrqm/s   每秒进行 merge 的读操作数目。即 delta(rmerge)/s
wrqm/s    每秒进行 merge 的写操作数目。即 delta(wmerge)/s
r/s       每秒完成的读 I/O 设备次数。即 delta(rio)/s
w/s       每秒完成的写 I/O 设备次数。即 delta(wio)/s
rsec/s    每秒读扇区数。即 delta(rsect)/s
wsec/s    每秒写扇区数。即 delta(wsect)/s
rkB/s     每秒读 K 字节数。是 rsect/s 的一半，因为每扇区大小为 512 字节。(需要计算)
wkB/s     每秒写 K 字节数。是 wsect/s 的一半。(需要计算)
avgrq-sz 平均每次设备 I/O 操作的数据大小 (扇区)。delta(rsect+wsect)/delta(rio+wio)
avgqu-sz 平均 I/O 队列长度。即 delta(aveq)/s/1000 (因为 aveq 的单位为毫秒)。
await     平均每次设备 I/O 操作的等待时间 (毫秒)。即 delta(ruse+wuse)/delta(rio+wio)
svctm     平均每次设备 I/O 操作的服务时间 (毫秒)。即 delta(use)/delta(rio+wio)
```

6.网络连接排查命令 netstat

1.常用选项：

```
netstat -tunlp   显示 tcp/udp 监听的端口和运行进程。如不使用-n 选项，会根据/etc/services 中的文件解析端口名称
/etc/services    该文件显示协议端口及对应的名称，可以自行更改
netstat -s       协议包数据统计
netstat -tan     显示所有的 TCP 链接，协助 tcp 状态机进行问题排查，最大 socket 65535 等
```

2.socket 设置：

对于 linux 操作系统来说，默认能够打开的 2^16=65536(0- 65535)端口，而系统默认设置的随机端口号只有 28232 个，可以通过以下文件查看：

```
$ cat /proc/sys/net/ipv4/ip_local_port_range
32768   61000
```

3.思路扩展：

对于一般的反向代理 server，除了自身服务监听端口外，可以使用的随机和后端 real server 链接的端口也是有限的的。需要使用上面的方法修改一下内核参数。而服务器对前端用户的响应是通过 socket 处理的。为了尽可能的提高并发访问量需要使用 ulimit 命令提升系统能够打开的最大文件描述符。socket 的端口是高速 TCP/IP 协议把请求发到这个 socket 处理程序上 socket.accept()来，然后生成的 socket 对象会记住具体是哪个客户端发来的请求。所以在系统性能可能打开最大的文件描述符的范围内，服务端可以用这一个端口服务无数的客户端。

4.加速端口回收实现端口快速重用：常用方法可以自行 Google。

7.网络流量抓取工具 tcpdump

tcpdump 命令的使用较为灵活，一般可以基于源和目的协议/端口/IP 进行数据抓去，详细使用语法这里就不再介绍了。需要特别注意的是我们一般使用-c 选项指定抓取数据包的数量，-w 选项以 wireshark 能够理解的方式保存抓去的内容。

通过以上的一些命令及问题的处理思路，大致能够对一台服务器的使用状况做初步的判断。欢迎大家随时交流~~

当网络遭受攻击时，如何快速找出真凶？

> 作者：swanor 来源：http://swenzhao.blog.51cto.com/3451741/1568327

在多年的 IDC 机房维护过程中，服务器不定期的遭受各种人士的来访。来访的方式一般有两种：

1、DDOS 攻击

2、利用系统漏洞植入后台程序

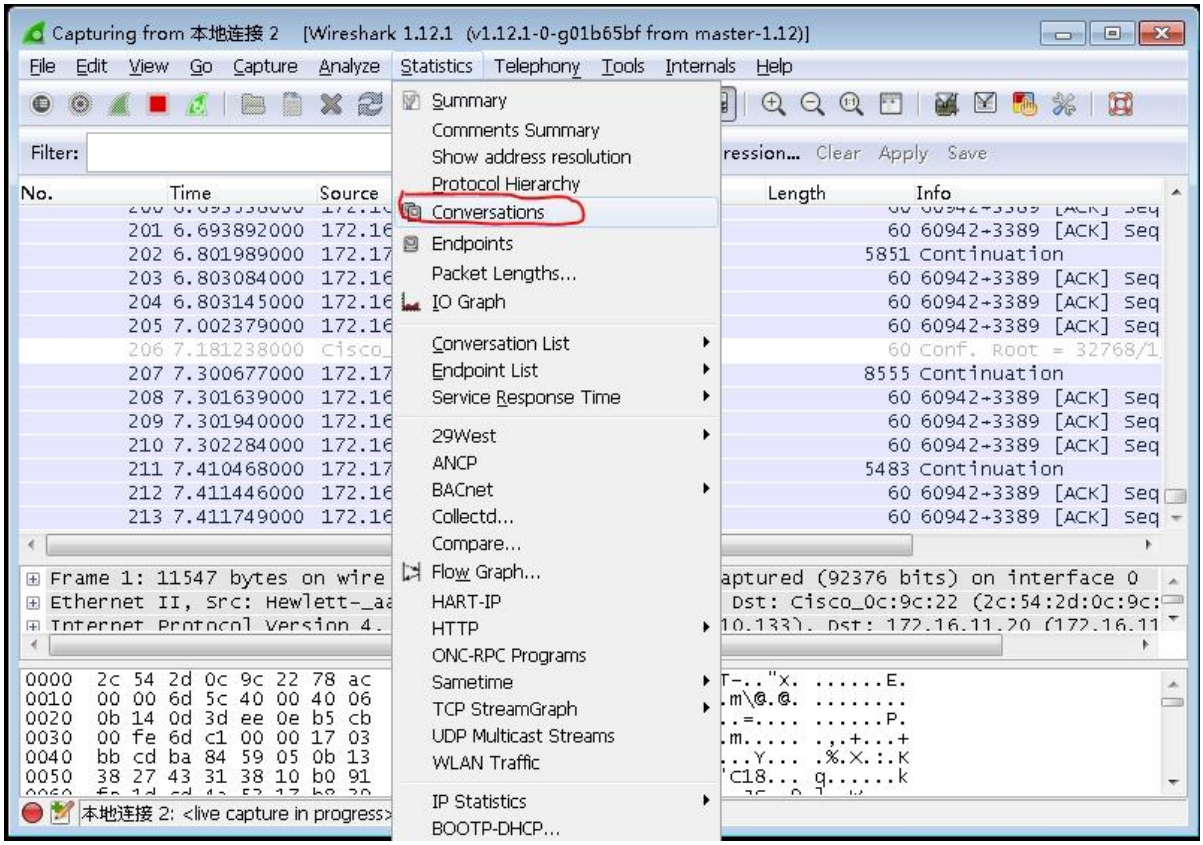
对于第一种情况，危害比较大，直接把入口堵死。但对服务器本身的伤害比较小。但第二种就很严重了，如果他入侵了 web 服务器，就可以直接访问后台数据库。一定要谨慎。

如果当遭受攻击，而你的整个防护设备中又没有入侵检测和流量分析设备。如何找出肇事者呢？这个时候使用开源的工具也可以。

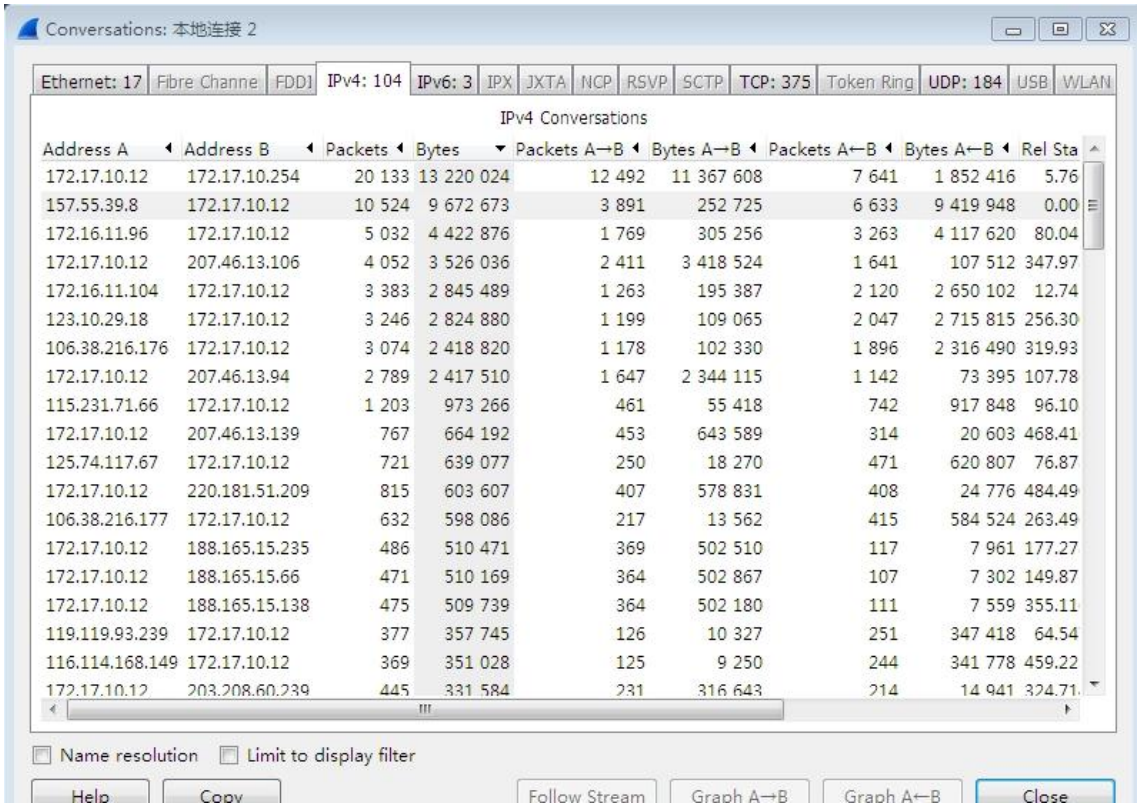
方法如下：

在连接到防火墙内网口的核心交换机端口上启用 SPAN（端口镜像），在目的端口上使用 wireshark 进行数据分析。很多人只知道 wireshark 可以抓包，可以看到数据包里面的内容。其实它还有一个很重要的功能就是以会话的方式统计数据流量。并进行排名。

至于交换机上的 SPAN 配置我这里就说明了。大家可以查找网上资料。下面是在 wireshark 上如何查看：



当然你先得开始数据包捕获功能。打开“conversation”后，可以看到如下，点击 IPV4，最上面的按钮是排序用的。可以看出，当前哪些人正在访问你。source 和 destination 都有了。



如果 source 的数据量过大，基本就可以断定是肇事者。可以使用很多方法进行过滤了。

让“开源”承包整个 IT 系统----你造吗？Linux 软件一直很拼的。

作者：chenjunjulian 来源：http://2574478.blog.51cto.com/2564478/1568446

哥浸淫企业 IT 架构设计这么多年，“私人定制”的设计产品五花八门，既有过“高大上”型的大品牌会战，也有过“屌丝”型的免费开源混搭。这次就跟大家来聊聊我最近的一次铁（yi）公（mao）鸡（bu）式（ba）企业 IT 系统搭建项目。

一. 需求分析：

用户企业是师妹注册刚开的一家文化传媒公司。硬件环境是两台大公司淘汰下来的服务器和一下网络设备。对内要有员工上网，文件交流、存储、打印，邮件收发；对外要有官网、论坛、会员博客，视频文件上传/下载等；远程要有员工在企业外访问内网资源。她的心很大，时髦的 IT 服务她都想到了。哎。既然是“师妹”相求，又是刚起步，你懂的了。哥恨不得达到“既要马儿好又要马儿不吃草”的效果。

二. 概要设计：

哥一直身处外企，养成了尊崇标准的习惯。所以这次设计还是沿用标准化的节奏，所有涉及到的开源软件都挑选的通过并持有 GPL（GNU General Public License 通用公共许可证）的。此设计的整体功能服务器划分为：

文件共享和打印服务器(Samba)，

反向代理（Web）服务器（Squid3.0+Apache+LNMP+Keepalived），

邮件服务器

（Postfix+Courier-IMAP+maildrop+Extmail+Extman+Amavisd-new+SpamAssassin+Clamav+Apache+phpMyAdmin+MySQL），

上网（正向）代理服务器（Squid3.0+Clamav+IPTables），

BBS/blog/FTP 服务器（Discuz!+X-Space+ProFTPd+MySQL），

远程接入服务器（PPTPD）。

三. 细节说明：

虚拟机：VirtualBox 4.3 改版本支持虚拟机的克隆和 SATA 硬盘的热插拔等特性。

操作系统：Linux 哪家强？当然是 CentOS 7。其 7 版本已将内核更新至 3.10.0。CentOS 高稳定性的服务器的上选。

文件共享和打印服务器：用 Samba 解决异构平台中数据共享与传输问题，值得一提的是如果企业日后发达了（希望师妹有朝一日变成“白富美”）升级或转换到微软架构，Samba 同样可以加入到活动目录（AD）中，实现单一登录访问（SSO）。

反向代理（Web）服务器：

方案采用的是 LNMP 架构，其中 Nginx 用作并实现负载均衡；Squid 用作并实现反向代理；Apache 用作并实现 Web 服务的发布。

除了 Nginx 本身，还要安装 ngx_cache_purge（缓存服务）。

而除了 Apache 本身，还要安装 PHP、PHPMyAdmin、php-memcache、eAccelerator、libmcrpt（php 扩展）、cmake（MySQL 的编译工具）、CoreSeek（MySQL 和 PHP 的中午检索）、多文件类型支持模块（包括 gb, libvpx, tiff, libpng, freetype, jpgsrc）。

另外，Apache 的安全工具：mod_clamav（防病毒），Ddos（防止 DDos 攻击）和 ModSecurity（应用防火墙）、以及验证模块（Sasl 和 Authlib）。

而数据库则选用 MySQL，因为它相对于 PostgreSQL 来说更流行更多支持和完善的文档资料。

邮件服务器：

Postfix：利用 SMTP 协议提供邮件发送服务。

POP3/IMAP 服务：Courier-IMAP：提供多种用户认证模块和方式、支持虚拟邮箱、支持共享目录并且可以限制 IMAP 同事登录的数目以及同一 IP 地址登录数，还能有效应对 DoS 攻击。另外 Dovecot 作为可选，因为其安全性比价出众，且一般 Linux 系统自带。

Maildrop 是必要的邮件投递代理，负责将邮件转发到用户邮箱。

SmapAssassin 是利用 Perl 对邮件内容来进行文字分析以达到过滤垃圾邮件的目的,它对邮件各种特征进行评分方式，总分高于门限值就视为垃圾邮件。

MailScanner 可以对 Postfix、SpamAssassin 和 ClamAV 进行总管并协调各模块之间的工作。但是个人觉得其必须监视 Postfix 的 Hold 队列，此“暴力”做法实在坑爹，因此我采用 Amavisd-new 对邮件病毒进行扫描。

那么对邮件系统、邮箱设置等综合管理我采用由 Perl 语言编写的 ExtMail，它具有面向大容量 ISP 级别应用和高性能。

WebMail：SquirrelMail，不使用任何 JavaScript 代码，纯粹用 PHP 所开发，从而兼容各种浏览器。它有支持增强的 MIME、地址簿、文件夹操作等功能。且一般 Linux 系统自带。

远程接入服务器：方案设计上采用的是构建 VPN 网络让远程用户拨入系统内部。选择 PPTPD 的原因是一般人员用的是 Windows 操作系统，而 Windows 客户端默认就支持 PPTP 方式的 VPN，Windows 用户不需要额外安装客户端软件而只需简单配置便可。

四.运维设计

妹纸的需求在功能上已满足了。那么问题来了，如果系统运行没一段时间就遇到问题或攻击，学妹是否会抱怨此等“终身大事”相托，却搞得如此脆弱，“人与人之间的基本信任都没有”。哥自认为是一个负责任的网络设计师，为了系统在实际使用中的平稳运营，我前瞻性的多走一步，设计了方便运维的模块。

文件存储与备份：考虑到文件服务器硬件配置且为二手，同时估计 IT 人员是刚毕业的“小鲜肉”，哥设计上采用的是 FreeNAS 软件的 NAS 架构，备份介质则用平时的移动硬盘便可。

网络管理 :网管软件采用的是 Nagios 的“分布-集中”模式对网络及应用的各种特征进行自定义监控，且有友好的 Web 界面和报表功能。

集中配置管理：我采用的是 C/S 结构的自动化运维工具 Puppet。

五. 后记

设计完成，学长只能帮你到这里了。之后，我协同其单位的 IT 专业小伙伴联合实施，历时 3 个多月，系统搭建完成并上线，除了 BBS/blog/FTP 服务器尚未投入使用外（不是臣妾做不到，而是师妹的合伙人—一位女汉子说这些锦上添花的功能等后期发展壮大了在逐步推出），一切运营正常且效果还不错。师妹夸我有“业界良心”，在其公司启动资金有限的情况下，实现了“多快好省”。我得意的笑并情不自禁的哼起了自己一直唾弃却总是不由得唱起的“你是我的小啊小苹果。。。。。。”

Linux 下高效数据恢复软件 extundelete 应用实战

作者：高俊峰 来源：http://ixdba.blog.51cto.com/2895551/1566856

作为一名运维人员，保证数据的安全是根本职责，所以在维护系统的时候，要慎之又慎，但是有时难免会出现数据被误删除的情况，在这个时候改如何快速、有效地恢复数据呢？本文我们就来介绍一下 Linux 系统下常用的几个数据恢复工具。

一、如何使用“rm -rf”命令

在 Linux 系统下，通过命令“rm -rf”可以将任何数据直接从硬盘删除，并且没有任何提示，同时 Linux 下也没有与 Windows 下回收站类似的功能，也就意味着，数据在删除后通过常规的手段是无法恢复的，因此使用这个命令要非常慎重。在使用 rm 命令的时候，比较稳妥的方法是把命令参数放到后面，这样有一个提醒的作用。其实还有一个方法，那就是将要删除的东西通过 mv 命令移动到系统下的/tmp 目录下，然后写个脚本定期执行清除操作，这样做可以在一定程度上降低误删除数据的危险性。

其实保证数据安全最好的方法是做好备份，虽然备份不是万能的，但是没有备份是万万不行的。任何数据恢复工具都有一定局限性，都不能保证完整地恢复出所有数据，因此，把备份作为核心，把数据恢复工具作为辅助是运维人员必须坚持的一个准则。

二、extundelete 与 ext3grep 的异同

在 Linux 下，基于开源的数据恢复工具有很多，常见的有 debugfs、R-Linux、ext3grep、extundelete 等，比较常用的有 ext3grep 和 extundelete，这两个工具的恢复原理基本一样，只是 extundelete 功能更加强大，本文重点介绍 extundelete 的使用。

三、extundelete 的恢复原理

在介绍使用 extundelete 进行恢复数据之前，简单介绍下关于 inode 的知识。在 Linux 下可以通过“ls –id”命令来查看某个文件或者目录的 inode 值，例如查看根目录的 inode 值，可以输入：

```
1[root@cloud1 ~]# ls -id /
```

22 /

由此可知，根目录的 inode 值为 2。

在利用 extundelete 恢复文件时并不依赖特定文件格式，首先 extundelete 会通过文件系统的 inode 信息(根目录的 inode 一般为 2)来获得当前文件系统下所有文件的信息，包括存在的和已经删除的文件，这些信息包括文件名和 inode。然后利用 inode 信息结合日志去查询该 inode 所在的 block 位置，包括直接块，间接块等信息。最后利用 dd 命令将这些信息备份出来，从而恢复数据文件。

四、 安装 extundelete

extundelete 的官方网站是 http://extundelete.sourceforge.net/ ，其目前的稳定版本是 extundelete-0.2.4。，在安装 extundelete 之前需要安装 e2fsprogs 和 e2fsprogs-libs 两个依赖包。e2fsprogs 和 e2fsprogs-libs 安装非常简单，这里不做介绍。下面是 extundelete 的编译安装过程：

真实案例：网站遭遇 DOS 攻击

作者：李晨光 来源：http://chenguang.blog.51cto.com/350944/1565327

一、事件背景

长假对于 IT 人员来 说是个短暂的休整时期，可 IT 系统却一时也不能停，越是节假日，越可能出大问题，下面要讲述的就是一起遭受 DOS 攻击的案例。

春节长假刚过完，小李公司的 Web 服务器就出了故障。下午 1 点，吃完饭回来，小李习惯性的检查了 Web 服务器。Web 服务器的流量监控系统显示下行的红色曲线，与此同时收到了邮件报警，可以判断服务器出现了状况。

根据上述问题，小李马上开始核查 Web 服务器的日志，尝试发现一些关于引起中断的线索。正当查询线索过程中，部门经理告诉小李，他已经接到客户的投诉电话，说无法访问他们的网站。

在 Web 服务器的日志文件中没有发现任何可疑之处，因此接下来小李仔细查看了防火墙日志和路由器日志。打印出了那台服务器出问题时的记录，并过滤掉正常的流量，保留下可疑的记录。表 1 显示了打印出来的结果。

表 1 防火墙日志统计

源 IP 地址	目的 IP 地址	源端口	目的端口	协议
172.16.45.2	192.168.0.175	7843	7	17
10.18.18.18	192.168.0.175	**19**	7	17
10.168.45.3	192.168.0.175	34511	7	17
10.18.18.18	192.168.0.175	19	7	17
192.168.89.111	192.168.0.175	1783	7	17
10.18.18.18	192.168.0.175	19	7	17

10.231.76.8	192.168.0.175	29589	7	17
192.168.15.12	192.168.0.175	17330	7	17
10.18.18.18	192.168.0.175	19	7	17
172.16.43.131	192.168.0.175	8935	7	17
10.23.67.9	192.168.0.175	22387	7	17
10.18.18.18	192.168.0.175	19	7	17
192.168.57.2	192.168.0.175	6588	7	17
172.16.87.11	192.168.0.175	21453	7	17
10.18.18.18	192.168.0.175	19	7	17
10.34.67.89	192.168.0.175	45987	7	17
10.65.34.54	192.168.0.175	65212	7	17
192.168.25.6	192.168.0.175	52967	7	17
172.16.56.15	192.168.0.175	8745	7	17
10.18.18.18	192.168.0.175	19	7	17

他在路由器日志上做了同样的工作并打印出了看上去异常的记录。在表 5-1 中是网站遭受攻击期间，经过规整化处理后的路由器日志信息。

为了获取更多信息，小李接着查看了路由器中 NetFlow 综合统计信息，详情如下：

```
Router1#sh ip cache flow
IP packet size distribution (567238991 total packets):
1-32   64   96  128  160  192  224  256  288  320  352  384  416  448
.000 .984 .002 .002 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000
 480  512  544  576 1024 1536 2048 2560 3072 3584 4096 4608
.000 .000 .002 .008 .000 .002 .000 .000 .000 .000 .000 .000
IP Flow Switching Cache, 7823134 bytes
4799 active, 117234 inactive, 1237463904 added
702311287 ager polls, 0 flow alloc failures
Active flows timeout in 30 minutes
Inactive flows timeout in 15 seconds
Last clearing of statistics never
```

Protocol	Total Flows	Flows /Sec	Packets /Flow	Bytes /Pkt	Packets /Sec	Active(Sec) /Flow	Idle(Sec) /Flow
TCP-Telnet	22943	0.0	1	45	0.0	0.1	11.7
TCP-FTP	134820	0.0	1	47	0.0	2.4	13.7
TCP-FTPD	1983	0.0	1	40	0.0	0.2	11.3
TCP-WWW	3563	0.2	1	38	1.5	0.1	3.2
TCP-SMTP	7682	0.0	1	42	0.0	1.0	12.2
TCP-X	1892	0.0	1	40	0.0	0.6	11.2
TCP-BGP	1782	0.0	1	40	0.0	0.2	11.5
TCP-NNTP	2906	0.0	1	40	0.0	0.1	11.2
TCP-Frag	108	0.0	2	26	0.0	1.4	15.7
TCP-other	**4992871**	**0.1**	**1**	**40**	**65.5**	**0.4**	**28.7**
UDP-DNS	10345	0.0	1	54	0.0	0.9	18.0
UDP-NTP	629	0.0	1	41	0.0	9.5	17.8
UDP-TFTP	621	0.0	2	40	0.0	11.9	17.1
UDP-Frag	25	0.0	1	34	0.0	261.4	13.7
UDP-other	**182921340**	**39.2**	**1**	**41**	**48.1**	**0.5**	**12.0**
ICMP	1893457	0.0	10	674	0.5	7.9	13.7
IGMP	29	0.0	1569	1241	0.0	14.5	16.2
IP-other	7	0.0	21	64	0.0	17.7	16.9

为了有参考基准，他还打印了在 Web 服务器开始出现问题的前几周他保存的缓存数据（这些是正常状态的数据）。正常路由日志，如下所示：

```
router1#sh ip cache flow
IP packet size distribution (567238991 total packets):
1-32   64   96  128  160  192  224  256  288  320  352  384  416  448
.000 .002 .002 .002 .000 .000 .000 .000 .000 .000 .000 .000 .000 .000
 480  512  544  576 1024 1536 2048 2560 3072 3584 4096 4608
.000 .000 .002 .012 .006 .974 .000 .000 .000 .000 .000 .000
IP Flow Switching Cache, 529842 bytes
2092 active, 50378 inactive, 8924 added
32341 ager polls, 0 flow alloc failures
Active flows timeout in 30 minutes
Inactive flows timeout in 15 seconds
last clearing of statistics never
Protocol      Total   Flows   Packets Bytes  Packets Active(Sec) Idle(Sec)
---------      Flows   /Sec     /Flow   /Pkt     /Sec     /Flow      /Flow
TCP-Telnet     1243    0.0         1     12      0.0      0.1        1.7
TCP-FTP        3452    0.0         1     23      0.0      1.4        6.3
TCP-FTPD        775    0.0         1     12      0.0      0.2        2.3
TCP-WWW    32467905    1.2         1     49      1.5      0.1        5.9
TCP-SMTP       3532    0.0         1     31      0.0      1.0        8.1
TCP-X          1692    0.0         1     38      0.0      0.8        8.2
TCP-BGP         975    0.0         1     32      0.0      0.2        9.5
TCP-NNTP       1674    0.0         1     28      0.0      0.1        9.2
TCP-Frag        103    0.0         2     23      0.0      1.0       11.7
TCP-other    496268    0.1         1     41     62.2      0.5       34.2
UDP-DNS        1342    0.0         1     43      0.0      0.9       14.9
UDP-NTP         323    0.0         1     33      0.0     10.0       12.6
UDP-TFTP        278    0.0         2     26      0.0      8.9        9.1
UDP-Frag         21    0.0         1     29      0.0    189.5        8.2
UDP-other      5632    0.2         1    171      0.2      0.5        1.9
ICMP         245685    0.0        10    693      0.5      8.4       12.9
IGMP             21    0.0      1387    988      0.0      6.2       15.8
IP-other          7    0.0        16     64      0.0     18.0       12.3
```

IP packet size distribution 这个标题下的两行显示了数据包按大小范围分布的百分率。这里显示的内容表明：只有 2%的数据包的大小在 33～64 字节之间。

注意，网站的访问量直线下降。很明显，在这段时间没人能访问他的 Web 服务器。小李开始研究到底发生了什么，以及该如何尽快地修复故障。

二、疑难问答

1.小李的 Web 服务器到底发生了什么？可能的攻击类型是什么？

2.如果地址未伪装，那么小李如何才能追踪到攻击者？

3.如果地址伪装过，那么他怎样才能跟踪到攻击者？

三、事件推理

小李的 Web 服务器遭受到什么样的攻击呢？这一攻击是通过对回显端口（echo 端口号为 7），不断发送 UDP 数据包实现。攻击看似发自两个地方，可能是两个攻击者同时使用不同的工具。在任何情况下，超负荷的数据流都会拖垮 Web 服务器。然而攻击地址源不确定，不知道是攻击源本身是分布的，还是同一个真实地址伪装出许多不同的虚假 IP 地址，这个问题比较难判断。假如源 IP 地址不是伪装的，则可以咨询 ARINI 美国 Internet 号码注册处，从它的“Whois”数据库查出这个入侵 IP 地址属于哪个网络。接下来只需联系那个网络的管理员就可以得到进一步的信息不过这对 DOS 攻击不太可能。

假如源地址是伪装的，追踪这个攻击者就麻烦得多。若使用的是 Cisco 路由器，则还需查询 NetFlow 高速缓存。但是为了追踪这个伪装的地址，必须查询每个路由器上的 NetFlow 缓存，才能确定流量进入了哪个接口，然后通过这些路由器接口，逐个往回追踪，直至找到那个 IP 地址源。然而这样做是非常难的，因为在 Web Server 和攻击者的发起 PC 之间可能有许多路由器，而且属于不同的组织。另外，必须在攻击正在进行时做这些分析。如果不是由司法部门介入很难查到源头。

经过分析之后，将防火墙日志和路由器日志里的信息关联起来，发现了一些有趣的相似性，如表 5-1 中粗黑体黑色标记处。攻击的目标显然是 Web 服务器（192.168.0.175，端口为 UDP 7。这看起来很像拒绝服务攻击(但还不能确定，因为攻击的源 IP 地址分布随机）。地址看起来是随机的，只有一个源地址固定不变，其源端口号也没变。这很有趣。他接着又将注意力集中到路由器日志上。

他发现，攻击发生时路由器日志上有大量的 64 字节的数据包，而此时 Web 服务器日志上没有任何问题。他还发现，案发时路由器日志里还有大量的“UDP-other”数据包，而 Web 服务器日志也一切正常。这种现象与基于 UDP 的拒绝服务攻击的假设还是很相符的。

此时 ,可假设攻击者正是用许多小的 UDP 数据包对 Web 服务器的回显(echo 7)端口进行洪泛式攻击，因此小李他们的下一步任务就是阻止这一攻击行为。首先，小李在路由器上堵截攻击。快速地为路由器设置了一个过滤规则。因为源地址的来源随机，他们认为很难用限制某个地址或某一块范围的地址来阻止攻击，因此决定禁止所有发给 192.168.0.175 的 UDP 包。这种做法会使服务器丧失某些功能，如 DNS，但至少能让 Web 服务器正常工作。

路由器最初的临时 DOS 访问控制链表(ACL)如下：

access-list 121 remark Temporary block DoS attack on web server 192.168.0.175

access-list 105 deny udp any host 192.168.0.175

access-list 105 permit ip any any

这样的做法为 Web 服务器减轻了负担，但攻击仍能到达 Web，在一定程度上降低了网络性能。 那么下一步工作是联系上游带宽提供商，想请他们暂时限制所有在小李的网站端口 7 上的 UDP 进入流量，这样做会显著降低网络上到服务器的流量。

四 、针对措施

对于预防及缓解这种带宽相关的 DOS 攻击并没有什么灵丹妙药。本质上，这是一种“粗管子打败细管子”的攻击。攻击者能“指使”更多带宽，有时甚至是巨大的带宽，就能击溃带宽不够的网络。在这种情况下，预防和缓解应相辅相成。

有许多方法可以使攻击更难发生，或者在攻击发生时减小其影响，具体如下：

网络入口过滤网络服务提供商应在他的下游网络上设置入口过滤，以防止假信息包进入网络。这将防止攻击者伪装 IP 地址，从而易于追踪。网络流量过滤软件过滤掉网络不需要的流量总是不会错的。这还能防止 DOS 攻击，但为了达到效果，这些过滤器应尽量设置在网络上游。

网络流量速率限制。一些路由器有流量速率的最高限制。这些限制条款将加强带宽策略，并允许一个给定类型的网络流量匹配有限的带宽。这一措施也能预先缓解正在进行的攻击。

入侵检测系统和主机监听工具。IDS 能警告网络管理员攻击的发生时间 ,以及攻击者使用的攻击工具，这将能协助阻止攻击。主机监听工具能警告管理员系统中是否出现 DOS 工具

单点传送 RPF（Reverse Path Forwarding），这是 CEF（路由器的 Cisco Express Forwarding 功能简称）用于检查在接口收到的数据包的另一特性。如果源 IP 地址 CEF 表上不具有与指向接收数据包时的接口一致的路由 ,路由器就会丢掉这个数据包。丢弃 RPF 的妙处在于 ,它阻止了所有伪装源 IP 地址的攻击。

1）检测 DOS 攻击

利用主机监测系统和 IDS 系统联合分析，可以很快发现问题，例如通过 EtherApe 工具(一款监视连接的开源工具)，当然，利用 Sniffer Pro 以及科莱网络分析工具可以达到同样效果。Sniffer 能实时显示网络连接情况，如果遇到 DOS 攻击,从它内部密密麻麻的连线，以及 IP 地址就能初步判定攻击类型，这时可以采用 Ossim 系统中的流量监控软件例如 Ntop，以及 IDS 系统来仔细判断。后两者将在《Unix/Linux 网络日志分析与流量监控》一书中详细讲解。最快捷的方式还是命令行，我们输入以下命令：

netstat -an|grep SYN_RECV|wc –l

通过结果可以发现网络中存在大量 TCP 同步数据包，而成功建立 TCP 连接的却寥寥无几，根据 TCP 三次握手原理分析可知，这肯定不是正常现象，网络肯定存在问题，需要进一步查实，如果数值很高，例如达到上千数值，那么很有可能是受到了攻击。如图 1 所示。

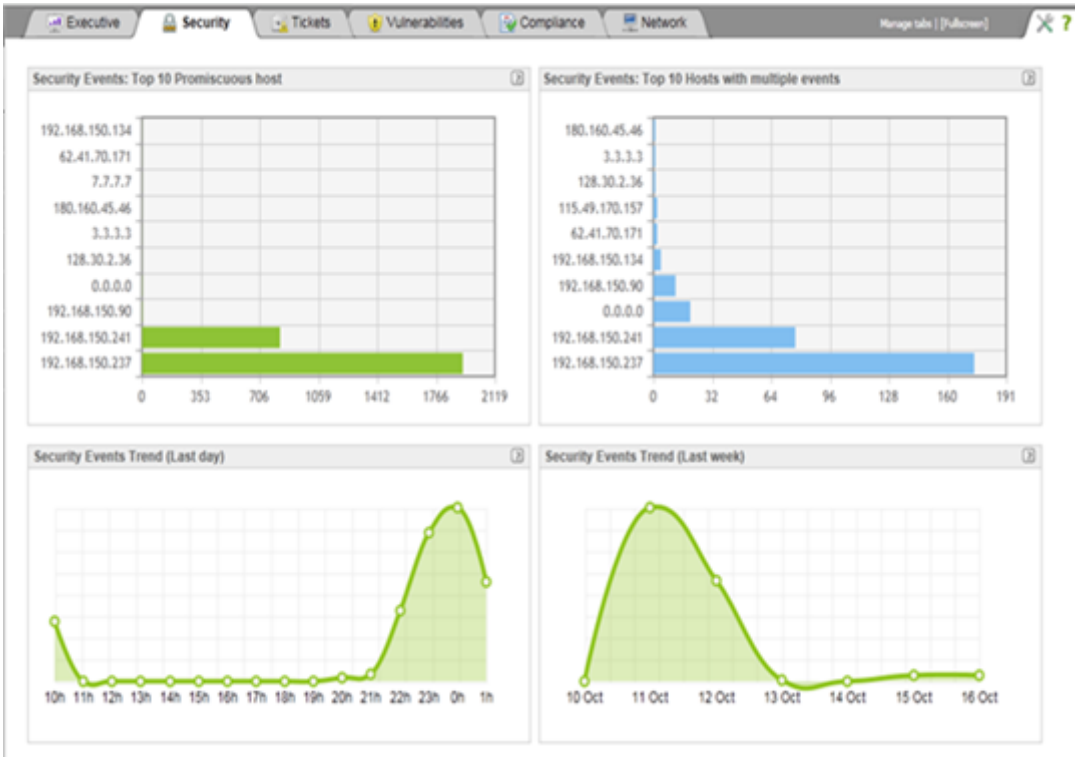


图 1 Ossim 发现 DOS 攻击

在图 1 中 OSSIM 系统中的 Snort 检测到 DOS 攻击并以图形方式显示出大量告警信息。例如，某网站在受到 DOS 攻击时 TCP 连接如下：

```
tcp        0      0 192.168.150.239:80      7.7.162.71:77           SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.248.20:77           SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.64.105:77           SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.23.55:77            SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.202.102:77          SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.196.200:77          SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.157.236:77          SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.98.114:77           SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.58.151:77           SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.255.202:77          SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.244.130:77          SYN_RECV
tcp        0      0 192.168.150.239:80      7.7.243.112:77          SYN_RECV
```

我们统计“SYN_RECV”状态的数量，命令如下：

#netstat –na |grep SYN_RECV |wc –l

1989

这么大数值，在配合上面 5-1 图形可以判断网站受到 DOS 攻击。

小技巧：还可以用下面的 Shell 命令，显示哪个 IP 连接最多。

#netstat -nta |awk ‘{print $5}’ |cut –d:f1 |sort|uniq –c |sort –n

1 192.168.150.10

2 192.168.150.20

… …

1987 192.168.150.200

这条命令得到的信息更详细。数值达到 1989，有近两千条，这明显说明受到了 DOS 攻击。 这时我们利用 Wireshark 工具进行数据包解码可以法相更多问题，当前通讯全都是采用 TCP 协议，查看 TCP 标志发送所有的数据包均为 SYN 置 1，即 TCP 同步请求数据包，而这些数据包往往指向同一个 IP 地址。至此可以验证上面的判断：这台主机遭受到 DOS 攻击，而攻击方式为 SYN Flood 攻击。

五、疑难解答

1.小李的服务器遭到了 DOS 攻击，攻击是通过对端口 7 不断发送小的 UDP 数据包实现的。这次攻击看起来源自两个地方，很可能是两个攻击者使用不同的工具。大量的数据流很快拖垮 Web 服务器。难点在于攻击地址源不确定，攻击源本身是分布的，还是同一个地址伪装出的许多不同 IP 地址不好确定。

2.假设地址不是伪装的，小李查询 ARIN，从它的 Whois 数据库中查出这个入侵 IP 地址属于哪个网络。

3.如果 IP 地址是伪装的，这种追踪比较麻烦，需要查询每台路由器上的 NetFlow 数据，才能确定流量进出在哪些接口，然后对这些路由器一次一个接口的往回逐跳追踪查询，直到找到发起的 IP 地址源。但是这样做涉及多个 AS（自治系统），如果在国内寻找其攻击源头

的过程往往涉及很多运营商 ,以及司法机关 ,工作量和时间都会延长 ,如果涉及跨国追查工作就更加复杂。最困难的是必须在攻击期间才能做准确分析，一旦攻击结束就只好去日志系统里查询了。

看了上面的实际案例我们也了解到，许多 DoS 攻击都很难应对，因为搞破坏的主机所发出的请求都是完全合法、符合标准的，只是数量太大。我们可以先在路由器上借助恰当的 ACL 阻断 ICMP echo 请求。

Router(config)#ip tcp intercept list 101

Router(config)#ip tcp intercept max-incomplete high 3500

Router(config)#ip tcp intercept max-incomplete low 3000

Router(config)#ip tcp intercept one-minute high 2500

Router(config)#ip tcp intercept one-minute low 2000

Router(config)#access-list 101 permit any any

如果能采用基于上下文的访问控制(Context Based Access Control,CBAC)，则可以用其超时和阈值设置应对 SYN 洪流和 UDP 垃圾洪流。例如：

Router(config)# ip inspect tcp synwait-time 20

Router(config)# ip inspect tcp idle-time 60

Router(config)# ip inspect udp idle-time 20

Router(config)# ip inspect max-incomplete high 400

Router(config)# ip inspect max-incomplete low 300

Router(config)# ip inspect one-minute high 600

Router(config)# ip inspect one-minute low 500

Router(config)# ip inspect tcp max-incomplete host 300 block-time 0

警告：建议不要同时使用 TCP 截获和 CBAC 防御功能，因为这可能导致路由器过载。

打开 Cisco 快速转发(Cisco Express Forwarding，CEF)功能可帮助路由器防御数据包为随机源地址的洪流。可以对调度程序做些设置，避免在洪流的冲击下路由器的 CPU 完全过载：

```
Router(config)#scheduler allocate 3000 1000
```

在配置之后，IOS 会用 3 秒的时间处理网络接口中断请求，之后用 1 秒执行其他任务。对于较早的系统，可能必须使用命令 scheduler interval<milliseconds>。

另一种方法是利用 Iptables 预防 DOS 脚本

```
#!/bin/bash
netstat -an|grep SYN_RECV|awk '{print$5}'|awk -F: '{print$1}'|sort|uniq -c|sort -rn|awk '{if ($1 >1)
print $2}'
for i in $(cat /tmp/dropip)
do
/sbin/iptables -A INPUT -s $i -j DROP
echo  “$i kill at `date`”  >>/var/log/ddos
done
```

该脚本会对处于 SYN_RECV 并且数量达到 5 个的 IP 做统计，并且把写到 Iptables 的 INPUT 链设置为拒绝。

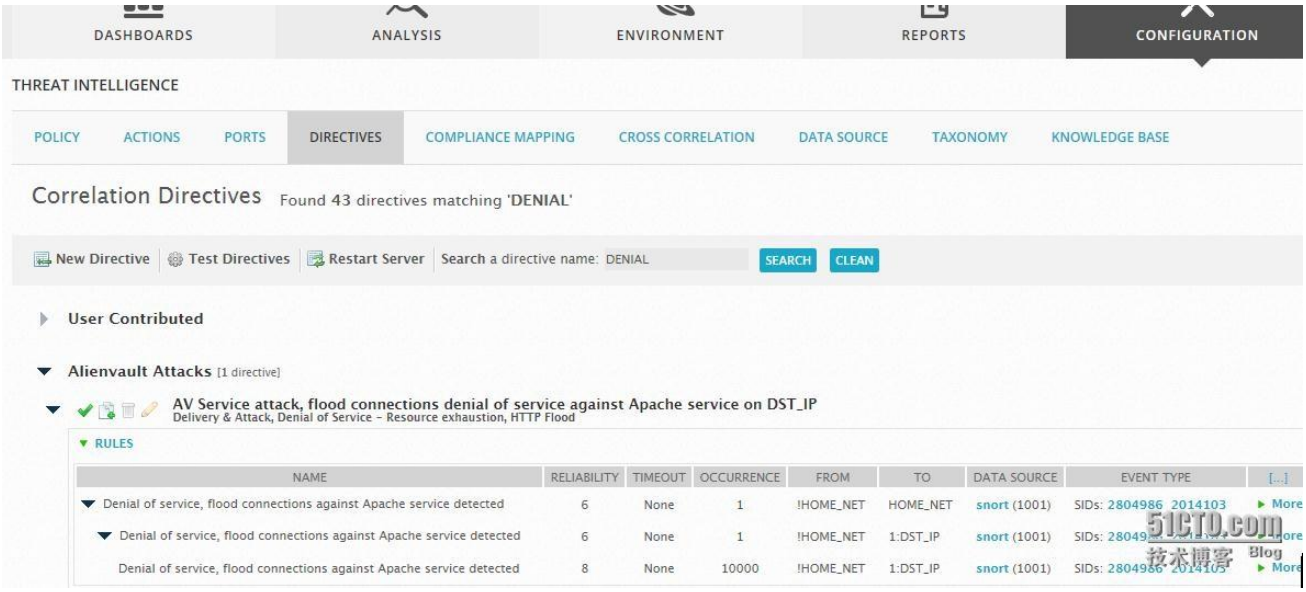
六、案例总结

无论是出于何种目的而发起更大规模攻击或其他目的 DOS/DDoS 攻击都必须重视。防范这种攻击的办法主要有及时打上来自厂商的补丁。同时，要关闭有漏洞的服务，或者用访问控制列表限制访问。常规的 DOS 攻击，特别是 DDOS 攻击更难防范。如果整个带宽都被 Ping 洪流耗尽，我们能做的就很有限了。针对 DOS 攻击，首先要分析它的攻击方式，是 ICMP Flood 、UDP Flood 和 SYN Flood 等流量攻击，

还是类似于 TCP Flood、CC 等方式，然后再寻找相对有效的应对策略。对于这种攻击可以采取下面介绍的几种方法：

1）.利用“蜜网”防护,加强对攻击工具和恶意样本的第一时间分析和响应。大规模部署蜜网设备以便追踪僵尸网络的动态,捕获恶意代码。部署网站运行监控设备,加强对网页挂马、访问重定向机制和域名解析的监控,切断恶意代码的主要感染途径。采用具备沙箱技术和各种脱壳技术的恶意代码自动化分析设备,加强对新型恶意代码的研究,提高研究的时效性。

2).利用 Ossim 系统提供的 Apache Dos 防护策略可以起到监控的作用。



3）.利用云计算和虚拟化等新技术平台，提高对新型攻击尤其是应用层攻击和低速率攻击的检测和防护的效率。国外已经有学者开始利用 Hadoop 平台进行 Http Get Flood 的检测算法研究。

4）.利用 IP 信誉机制。在信息安全防护的各个环节引入信誉机制,提高安全防护的效率和准确度。例如对应用软件和文件给予安全信誉评价，引导网络用户的下载行为，通过发布权威 IP 信誉信息，指导安全设备自动生成防护策略，详情见《Unix/linux 网络日志分析与流量监控》2.1 节。

5）.采用被动策略即购买大的带宽，也可以有效减缓 DDOS 攻击的危害。

6).构建分布式的系统 ,将自己的业务部署在多地机房 ,将各地区的访问分散到对应的机房 ,考虑部署 CDN，在重要 IDC 节点机房部署防火墙（例如 Cisco、Juniper 防火墙等）这样即使有攻击者进行 DOS 攻击，破坏范围可能也仅仅是其中的一个机房，不会对整个业务造成影响。

7).如果规模不大，机房条件一般，那可以考虑在系统中使用一些防 DDos 的小工具，如 DDoS Deflate，它的官网地址是 http://deflate.medialayer.com ,它是一款免费的用来防御和减轻 DDOS 攻击的脚本，通过系统内置的 netstat 命令，来监测跟踪创建大量网络连接的 IP 地址，在检测到某个结点超过预设的限制时，该程序会通过 APF 或 IPTABLES 禁止或阻挡这些 IP。当然此工具也仅仅是减轻，并不能全部防止攻击。

最后还要用不同供应商、不同 AS 路径并支持负载均衡功能的不止一条到因特网的连接，但这与应对消耗高带宽的常规 DOS/DDOS 洪流的要求还有差距。我们总是可以用 CAR（Committed Access Rate，承诺访问速率）或 NBAR(Network-Based Application Recognition，网络应用识别)来抛弃数据包或限制发动进攻的网络流速度，减轻路由器 CPU 的负担，减少对缓冲区和路由器之后的主机的占用。

谁动了我的文件？

作者：李晨光 来源：http://chenguang.blog.51cto.com/350944/1561081

一、事件背景

本文描述了 IT 经理小李在一起广告公司文件泄露的案件中，通过对交换机、服务器日志和邮件信头进行分析，利用多方面日志内容验证了他的推测，最后他将这些蛛丝马迹汇总起来，勾勒出了这次攻击事件的完整过程。大家在看完事件的描述后，是否知道在 FTP 和 SSH 日志中找到了什么线索？下面故事开始啦。

故事主人公小李在一家渲染农场（Render Farm）电影特效公司上班，前不久刚刚被提升为 IT 经理，这对于他来说是一件无比兴奋的事情。目前他们公司正在制作《某 电影》的特技效果，大家都为之共同努力工作。他每天早上必须喝上一杯咖啡。今天，他拿着咖啡向办公室走去。被小周和小王叫进会议室。接着，小王开始讲述事件的要点，不过没有提及任何一个同事的名字。然后，小王对小李说：“老大，有人将《某电影》的机密信息散布了出去，在某电影网站上发布了 1 分钟片长的电影片段。”小周对此非常重视，他让我们查出是谁干的（这些视频特效在昨天早晨刚完成后期制作）。

小李有点紧张，此刻他意识到已经发生的事情对于他们来说意味着什么。小周大声说道：“泄露出去的片段是观众最期望看到的内容，但现在已经公诸于众了！单单是一个镜头就能让公司直接经济损失达数几十万元！”小曹接着补充说，电影制作公司将不会再把自己的电影特效交给他们制作，除非他们将这件事情查个水落石出，并且能防止其再次发生。小李这才明白过来，他们不仅失去了这部电影的特效渲染工作，如果消息传开的话，他们将失去更多的机会。

二、了解业务流程

小李只是 IT 人员，并不熟悉动画渲染业务，他为了搞清楚公司业务流程，立刻询问了电影胶片制作的所有过程，从梦工厂收到电影制作公司的电影胶片，直到这些电影胶片运回到电影制作公司。小周一一叙述了整个过程，因为这些都是在他的监督之下完成的。制片公司将需要后期制作的电影胶片（需采用非线

性编辑的视频特效）存放在硬盘上，小王将硬盘上的内容复制到 RAID 阵列上，然后给后期制作小组发电子邮件，告诉他们可以取胶片。

后期制作小组的工作采取轮班工作方式，所以小周打算查出昨天是谁处理过某某电影的视频。团队完成后期制作，视频文件就被放在了服务器上的一个目录下。待硬盘中存储足够多的文件，小王才将硬盘上的文件送给电影制作公司。然后这些文件会被写入磁盘阵列上并离线保存，目前《某电影》视频内容还没有归档到阵列上。

三、公司内鬼所为？

调查工作进行到第二天清晨，还是没有得到有价值的线索。小李认为有必要和小王进行一次交谈，以便进一步了解技术细节。小李心想：“难道是小王把视频卖给影迷网站？他是那种人吗？”小李必须得弄个水落石出。

小王在公司创建之初就来工作，现已工作多年。他是后期制作团队的系统管理员。小王和小李之间联系不多，因为后期制作相对独立。小王再一次向小李解释了所有的过程，他愿意提供更多的技术细节，他们的磁盘阵列和 Linux 服务器之间采用直连方式，与该服务器相连的所有客户端也清一色使用 Linux 系统。所有的后期制作成员都使用 Web 浏览器来获取他们想要操作的文件，并且挑出他们正在处理的文件，也就是说不可能两个人同时操作同一个电影胶片。这些 Web 上的代码都是两年前由公司内部开发的，非常可靠。

小李从与小王的谈话中确信他不会是作案者。首先，他不会为了贪图眼前的利益而毁掉自己的前程；其次，小李非常赞赏他的业务能力。

小李回到了自己的办公室，思考着下一步该怎么办。这似乎并不像内部职员所为。公司内有着良好的企业文化，假设《某某电影》这部惊人之作能够家喻户晓的话，渲染农场公司也会因此迎来自己的辉煌。小李决定仔细研究一下网络拓扑图。公司的网络拓扑如图 1 所示，这是他的前任临走前留给他的，也许能够从中找到一些启示。

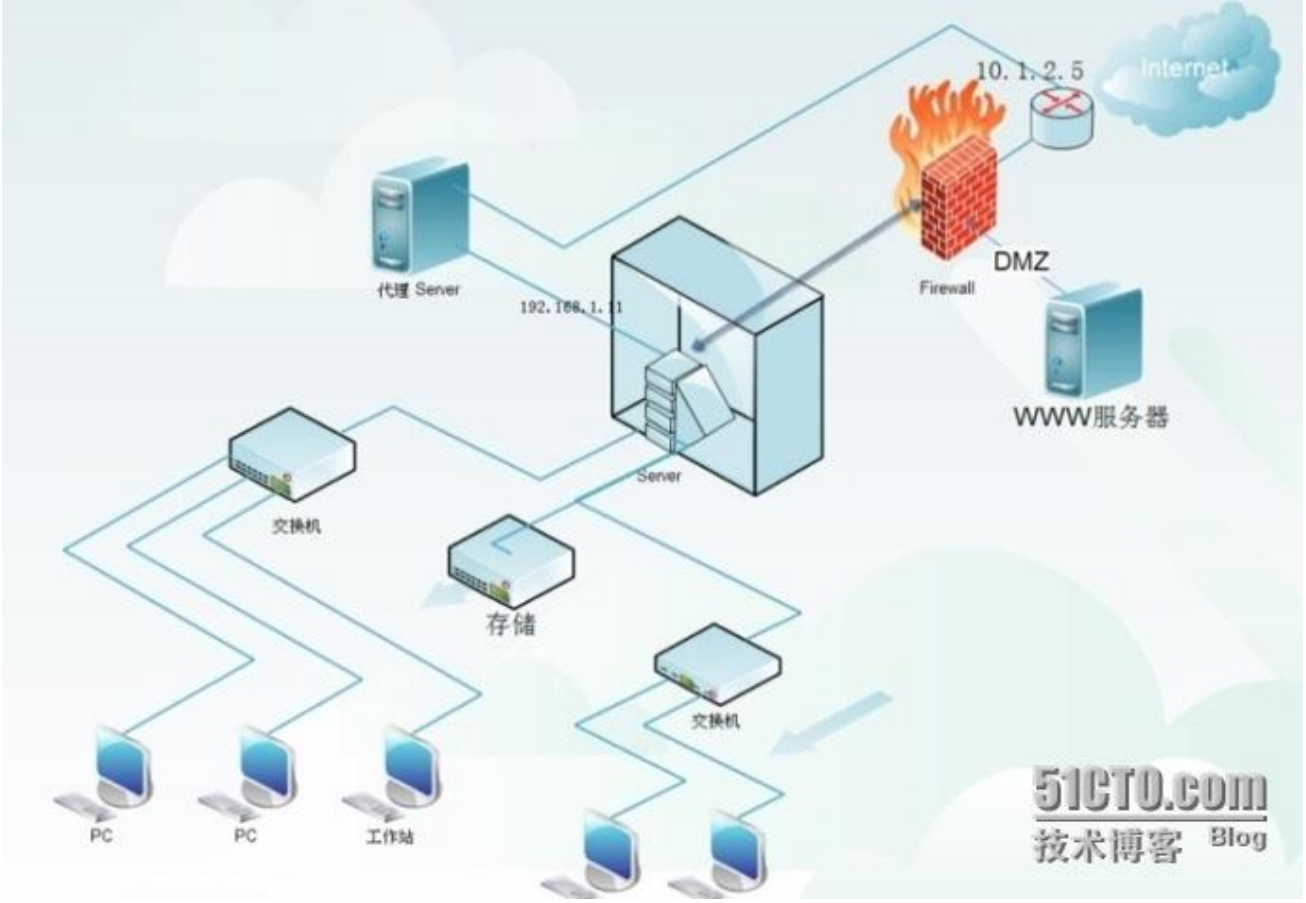


图 1 案例网络拓扑

这张网络拓扑图似乎并没有给小李太多帮助，局域网中只有几个 VLAN。公司内网和因特网之间也有防火墙、DMZ 区和代理服务器，一切看上去都很正常，调查工作陷入僵局。这时，小王来到小李的办公室说，小蒋是昨天最后一个调取某电影胶片文件的员工。小李立刻拿着记事本去找小蒋，打算一探究竟。

小蒋是公司的新员工，小李曾经见过他几次，但并没有和他谈过话。小蒋告诉小李他工作的整个过程：首先他将视频文件从服务器下载，然后对它进行编辑加工，接着就将修改过的文件再提交给服务器。小李询问上传下载的方法时，小蒋说是使用 FTP 下载。小李听到这条线索，他觉得这也许就是问题所在。于是，他接着问小蒋修改完文件后上传的时间。小蒋会议了一下说，“昨天是我太太生日，所以晚上下班比较准时，大约时间是在 5:15～5:30”。

四、取证分析

小李决定先找小王查看后期服务器上的 FTP 日志。小王很高兴事情有了新的进展，他帮助小李查询 FTP 日志文件，并登录了后期制作服务器。

```
# grep xiaojiang xferlog

Mon Sept 10 04:48:18 2010 1 1.example.com 147456 /var/ftp/pubinfo/bdsq/file2.jpg b_oa xiaojiang ftp 0*i
```

“好，这说明小蒋是正常上传文件的。但是之后会不会又有人调取过呢？”

```
#grep jer xferlog

Mon Sept 10 04:48:18 2010 1 1.example.com 147456 /completed/ hawk.avi b_oa Jer ftp 0*i
```

小李有点糊涂了。小蒋是正常上传文件的，在这之后再没人访问过，至少没人再通过 FTP 访问过。小李一头雾水地回到了自己的办公室。小王看到小李并拦住了他，连忙询问是否有新的发现。然而小李只能对他解释说发现了一些可疑之处，但还没有得到证实，小李又一次感到自己一整天都像是热锅上的蚂蚁。就在这时候，小周来到了小王的办公室。同样的事情发生了！ 急急忙忙说：“又有一部制作好的片头视频被发布在某电影网站上了”。看到经理这样的情形，小王和小李的心里砰砰直跳。小周说道视频文件是昨天晚上制作完成的,怎么这么快就泄露了呢？这时小李想到联系某电影网站的网管联系，看看是谁发布了这个视频。当和网管取得了联系后，从网管说这些信息来自一位自称是个 Tom 的人，他的电子邮件地址是 tom@yahoo.com.cn。

下面小李开始利用这封邮件的邮件头信息找到他的 IP 电子邮件证据认定的实例分析，他找到了下列邮件头信息：

```
Received: from web15604.mail.cnb.yahoo.com ( [ 202.165.102.x ] ) by SNT0 -MC3 -F14.Snt0.hotmail.com with Microsoft SMTPSVC（6.0.3790.4675）；Sat，24 Sep 2010 08:17:50 -0700

Received: from [122.246.51.2x] by web15604.mail.cnb.yahoo.com viaHTTP；Sat，24 Sep 2010 23:17:48 CST
```

```
X-Mailer:YahooMailWebService/0.8.114.317681
Message-ID: <1316877468.60773.YahooMail-Neo@web15604.mail.cnb.yahoo.com>
Date: Sat，24 Sep 2010 23:17:48 +0800（CST）
From: zhen tom@yahoo.com.cn
Reply -To: tom fei tom @yahoo.com.cn
Subject: test by webmail
To: =?utf-8?B?6LS56ZyH5a6H?= tom@hotmail.com
```

经过认真分析、反复核对，小李基本确定了他的 IP 地址,而且他利用 OSINT tools 工具箱中的 Maltego CE 工具输入该电子邮件地址，经过简单配置，系统开始搜索到有关这个邮箱更多的信息。小李来到小王的办公桌前，想看看是否能够找到一些其他的信息，也许会有些头绪。小李让小王再次检查一下 FTP 日志。

```
#grep apple1.avi xfelog
Mon Sept 10 04:48:18 2010 1 postprod 147456/completed/apple1.avi b_oa\lex ftp 0*i
```

同样，在工作人员上传完文件之后没有人再访问过这些文件。小李问小王是否还有其他的方法能够获取这些文件。小王解释说，这台主机设置了防火墙，只允许 21、22、80 端口通过，也就是只允许通过 SSH、FTP 和 Apache 三种服务访问。于是小李又让小王检查在这些文件上传 FTP 服务器之后的 SSH 日志文件。

```
Sep 10 17:24:58 postprod sshd[3211]:Accepted password for wanglei from 192.168.0.3 port 49172 ssh2
Sep 10 18:03:18 postprod sshd[3211]:Accepted password for wanglei from 192.168.0.3 port 49172 ssh2
Sep 10 22:13:38 postprod sshd[3211]:Accepted password for wanglei from 192.168.0.3 port 49172 ssh2
```

同样的结果，小李感到非常失落。现在的问题是，没有人访问过这些文件，那这些文件又是怎么泄漏出去的呢？接下来小王只好查看 Web 服务器日志文件，看看能否查到一点线索。

```
#grep hawk.avi /var/log/apache/

192.168.1.11--[10/Sep/2010:23:55:36 -0700] "GET /completed/hawk.avi HTTP/1.0"200 2323336
```

小王的眼睛亮了起来，小李也惊喜地张大了嘴巴，192.168.1.11 这个地址是公司内网地址，也许他们找到了“凶手”！他们发现了一个异常的 IP 地址，他之前从来没有见过这个 IP（192.168.1.11）地址。这个 IP 不属于 DHCP 范围之内，而属于一个静态服务器范围。小李问小王是否知道哪一台服务器使用这个 IP，小王不能确定。但这个 IP 一定不属于后期制作服务器群所在的 VLAN。小李决定再仔细查看一下 Web 服务器日志文件，这次主要是看一看这个可疑的 IP 地址：

```
# grep `192.168.1.11` /var/log/apache/

192.168.1.11--[10/Sep/2010:23:50:36 -0700] "GET /index.html HTTP/1.0"200 2326

192.168.1.11--[10/Sep/2010:23:55:36 -0700] "GET /completed/index.html HTTP/1.0" 200 2378

192.168.1.11--[10/Sep/2010:23:51:36 -0700] "GET /completed/movie-cab.avi HTTP/1.0 " 200
1242326

192.168.1.11--[10/Sep/2010:23:52:24 -0700] "GET /completed/hawk.avi HTTP/1.0" 200 2323336

192.168.1.11--[10/Sep/2010:23:55:36 -0700] "GET /completed/apple1.avi HTTP/1.0"200 642326

192.168.1.11--[10/Sep/2010:14:00:38 -0700] "GET /completed/pool.avi HTTP/1.0"200 662326

192.168.1.11--[10/Sep/2010:23:55:36 -0700] "GET /completed/less.avi HTTP/1.0"200 2552326
```

小李发现有一个人浏览了很多文件。在公司丢失更多的文件之前，小李必须查清楚到底发生了什么。小李告诉了小王新的进展，他为此很高兴，不过他希望小李能尽快找到事情的最终答案。小李回到了自己的座位上继续跟踪刚才日志上的可疑 IP。他感到非常兴奋，因为“嫌疑人”更近了，尽管他还并不清楚该从何开始。他认为追捕到这个 IP 地址的最佳办法是找到这个 IP 地址在物理上是从哪里连上网络的。要做到这一点，就要把该机器连接到交换机的端口以便和机器的 MAC 地址匹配起来。

五、遗忘的 Squid 服务器

小李首先 ping 这个 IP 地址，然后从 ARP 表中得到这台机器的 MAC 地址。

1）.追查非法使用端口

小李得到了重要的信息，他立刻远程登录到服务器连接的 Cisco 交换机上。经过几次尝试以后，有了重大的突破。

我们使用 ping 命令看看他机器网卡的 MAC 地址是多少。

```
Interface: 192.168.3.41 on Interface 0x1000003
Internet Address   Physical Address   Type
192.168.1.1          00-30-ab-04-26-dd     dynamic
192.168.1.11         00-0d-56-21-af-d6     dynamic
```

从本机的 ARP 缓存里看这个可疑 IP 地址的 MAC 地址为 00-0d-56-21-af-d6，登录交换机一看果然还是这个地址。

```
BJ-SW#show arp | in 192.168.1.11
Internet  192.168.1.11                3    000d.5621.afd6  ARPA    Vlan20
```

下一步，我们要知道他的机器是接到哪台交换机器上。

```
BJ-SW#show mac-address-table dynamic address 000d.5621.afd6
Unicast Entries
 vlan    mac address      type         protocols                port
-------+---------------+--------+--------------------+--------------------
  20     000d.5621.afd6    dynamic ip,ipx                  GigabitEthernet3/2
```

从结果看这是通过千兆端口连接。我们看看邻居（这是核心交换机器的二级级联交换机）

```
BJ-SW-419-1-4#sh cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater, P - Phone
Device ID        Local Intrfce      Holdtme     Capability  Platform  Port ID
```

```
SW-419-2-3     Gig 3/4          152          S I      WS-C3550-4Gig 0/2
SW-419-1-3     Gig 3/3          168          S I      WS-C3550-4Gig 0/2
SW-440-1-4     Gig 3/1          173          S I      WS-C3550-4Gig 0/2
SW-440-2-4     Gig 3/2          143          S I      WS-C3550-2Gig 0/2
```

终于找到它了，在 SW-440-2-4 Gig 3/2 143 SI WS-C3550-2Gig 0/2 上，下面我们直接登录到 SW-440-2-4 这台交换机，输入 MAC 查找。

```
SW-440-2-4#show mac-address-table dynamic address 000d.5621.afd6
          Mac Address Table
-------------------------------------------


Vlan    Mac Address         Type         Ports
----    -----------         --------     -----
  20    000d.5621.afd6      DYNAMIC      Fa0/23
Total Mac Addresses for this criterion: 1
```

然后根据综合布线时的跳线表就可直接到这台机器,接下来关闭该端口。

注意：作为管理人员，快速定位交换机端口，找出 IP 和 MAC 对应关系是必须掌握的一项技能，熟悉以上方法能为平时故障排除达到事半功倍的效果。

小李发现了这个系统就连在服务器交换机的第 23 个端口上，于是冲下大楼直奔小机房，迅速来到 Catalyst 交换机前找到第 23 端口，然后开始顺藤摸瓜。可杂乱繁多的网线，一看就头疼，查找问题花去了小李很多的时间。最后终于找到了连接的主机，小李发现，该主机内部有两块网卡。他从网线堆里爬出来，无奈地看着这台机子，机箱上面贴着一张发黄的旧标签，上面写着 Squid Proxy Server。这时小李立刻有种反胃的感觉，因为这台服务器至少 1 年多没有使用了，而且自从他升职后，这台服务器也确实没有使用过。

小李现在根本不能确定黑客到底是从哪儿入侵的，代理服务器又为他们的调查出了一个难题。小王倒是给小李提供了一些有用的线索，他把前任经理给他留下的服务器用户名及口令清单交给了小李。小李迅速回到自己的座位开始工作。他登录到 Squid 代理服务器上，希望这一次能有所发现。

六、疑点分析

小李迅速打开终端，用 SSH 登录到服务器，使用 root 用户和口令。成功登录系统了。小李很容易就查到 access.log 文件，现在他可以查出任何一个登录过该服务器的人。

```
squidbox#1s -1   /usr/local/squid/logs/access.log
-rw-rw-r-- 1 squid squid 2838159 Sep 11 03:25 access.log
```

这时问题出现了，由于 SQUID 服务器工作时间长，squid.log 的日志非常庞大，查个 IP 也不是容易的事，如何将 access.log 的 IP 提取出来呢？小李使用以下命令：

```
squidbox#awk   '{print$3;}'   access.log
```

小李看到了他最不愿看到的事---该文件最后一次修改的时间是今天的凌晨 3：00。现在应该看看这个文件：

```
squidbox# tail   /usr/local/squid/logs/access.log
892710014.016          14009          10.100.4x.5x          TCP_MISS/304          126          GET
http://192.168.2.3/completed/less.avi --
```

显然，在公司网络以外的人通过代理服务器进入过后期制作的 Web 服务器。通过日志文件，小李清楚地知道黑客在昨天夜里访问过 Hawk 和 Apple 的胶片。他非常希望这个黑客能够在今晚再次造访，以便抓个正着。

小李赶紧跑到小王的办公室，把这个消息告诉了他。找到了“凶手”，小王感到轻松了许多，同时希望能够找到更多关于这个黑客的信息。小李建议说，他们应该拔掉代理服务器外网的接口，防止黑客卷土重来。小王同意小李的建议，至少他们不能再泄露更多胶片文件。小李回到小机房，拔掉代理服务器外网口的网线（这段是连接互联网的）。然后回到自己的座位决定做一些侦察工作。他想继续跟踪这个黑客，

并且一定要给他一点儿颜色看看。因为此类入侵事件具有时效性，错过这个村就没这个店。这时小李决定架设一套蜜罐系统来诱捕黑客，以便获得更多的证据。

七、诱捕入侵者

由于 IDS 等网络设备昂贵，他们所在的公司无力更换新的安全设备，所以他设计了虚拟机下的蜜罐系统,下面的内容讲述了架设蜜罐系统的注意事项。

一般情况下，蜜网由蜜网网关、入侵检测系统及若干个蜜罐主机组成。其中蜜网网关是控制蜜网网络的枢纽，在网关上安装多种工具软件，对数据进行重定向、捕获、控制和分析处理，如 IPTables、Snort、SebekServer、Walleye 等。访问业务主机的流量不经过蜜网网关，而访问蜜罐的网络连接，都由重定向器引向蜜网，而攻击者往往无法察觉。本文描述的是在单一主机上模拟出整个蜜罐系统的解决方案，它是基于最新虚拟机软件 VMware 9 和虚拟蜜网技术，构建集网络攻击和防御于一体的网络安全平台。虚拟蜜网部分，除管理计算机外，其他都是基于虚拟机之上。安装虚拟机系统的宿主计算机（蜜网网关）的配置要求稍高，这样可更好地运行多个虚拟蜜罐操作系统。

接口描述：在虚拟蜜网网关中，有 3 个网络接口：

eth0 是面向业务网络的外部接口。

eth1 是面向多虚拟蜜罐系统的内部接口，Eth0 和 Eth1 在网桥模式，均无 IP 地址，数据包经过网关时 TTL 值也不会变化，也不会提供自身的 MAC 地址，因此蜜网网关对于攻击者是透明不可见的，入侵者不会识别出其所攻击的网络是一个蜜网系统。

eth2 是用作远程管理，具有实际 IP 地址，可把出入虚拟蜜罐系统的数据包及蜜网系统日志转发给一台作远程管理的主机。

架构详情如图 2 所示。

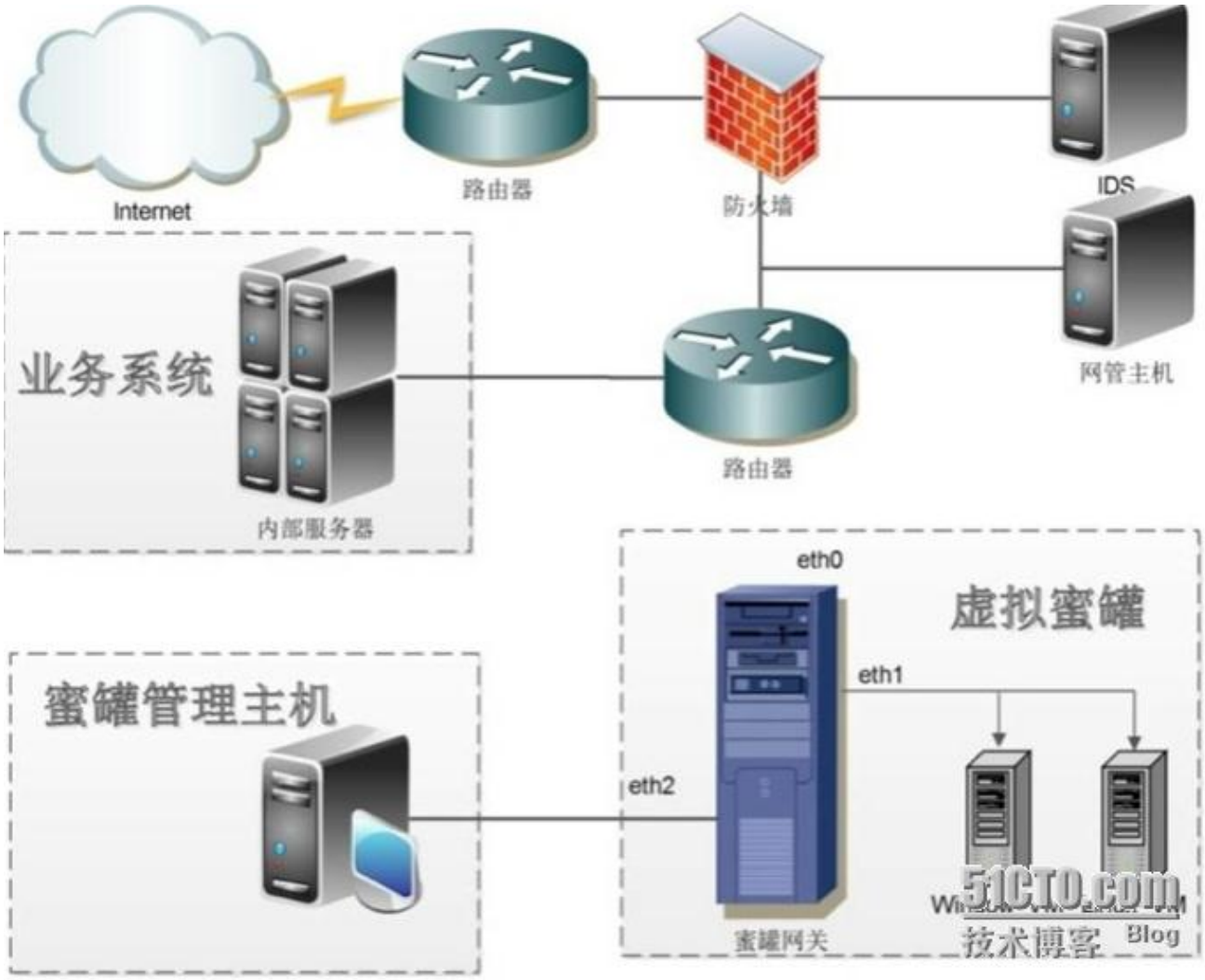


图 2 虚拟蜜罐架构图

架设好蜜网系统，就等神秘人再次造访，以便获取更多有价值的日志信息。小李先使用 nslookup 命令查看了该 IP 的 DNS 服务器的主机名、域名、地址。

```
#nslookup 10.100.4x.5x

Server: ns1.movie.com

Address: 10.1.1.11

Name:    chewie.someisp.ru

Address: 10.100.4x.5x
```

经过综合分析，比较邮件头信息和蜜罐系统中找到的 IP，完全吻合，这回发送者 IP 终于找到了，小李松了口气。下面就要通过电信找到这个人是在哪里拨号上网的，就能找到申请电话、住址及姓名。

八、预防措施

在这个案例中，小李并没有意识到网络中存在这样一个代理服务器，这简直糟透了。如果你管理一个网络，最好对网络进行安全审计。前任 IT 管理员留下的网络结构图中清楚地指明了代理服务器，但是因为服务器没有上线，所以没有引起小李的注意。既然代理服务器并没有真正使用，所以应该及时与网络断开。

开放式代理服务器的最大问题是访问控制列表的不合适的配置。对于 squid 代理服务器来说，合适的设置应该是下面这样：

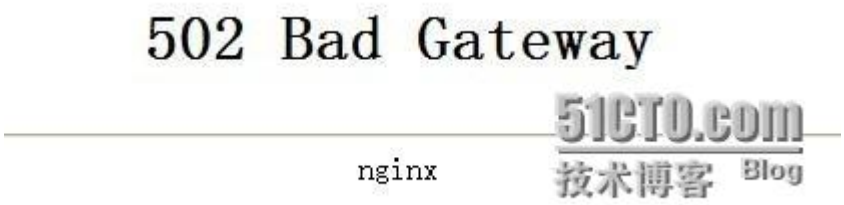
```
acl mynetwork src 192.168.1.0/255.255.255.0
http_access allow   mynetwork
http_access deny all
```

在这个案例中，小李采取了适当的补救方法。尽管在开始的时候他的方法有些不得当，但当他开始怀疑代理服务器的时候，他就从网络中断开了该服务器的物理连接。在服务器从物理上断开后，就可以开始细致的检查，而不用担心黑客会再次通过代理服务器入侵。同时小李还应该考虑检查所有的日志文件，这样才可能全面评估这次入侵造成的损失。如果在事发之前小李安装了集中日志收集系统，或者是 OSSIM 开源安全信息平台，则对于这些日志查询与分析将变得容易多了。

如何面对你—LNMP 高并发时 502

> 作者：zhang406 来源：http://zhangxylinux.blog.51cto.com/5041623/1563427

问题:最近的抢购有点火，到点抢购的时候网站就会出现 502 错误 顶不住消费者的压力。



伤。。。。。

之前 php-fpm 配置：

单个 php-fpm 实例，使用 socket 方式，内存 8G 静态方式，启动 php-fpm 进程数 300，具体参数如下

```
listen = /tmp/php-cgi.sock
#listen = 127.0.0.1:9000
listen.backlog = 2048
listen.allowed_clients = 127.0.0.1
pm = static
pm.max_children = 300
pm.start_servers = 50
pm.min_spare_servers = 30
pm.max_spare_servers = 250
request_terminate_timeout = 0
```

由于架构，代码等原因，单台几百并发就出现 502 错误。

初步解决：各种相关优化，

增大 pm.max_children 为 400

nginx 和 fpm 添加了 listen.backlog = 2048

最大打开文件句柄数 65535

/etc/sysctl.conf 都进行了微调 ,高并发时 nginx 发起的连接数 ,远远超过了 php-fpm 所能处理的数目，

导致端口（或 socket）频繁被锁，造成堵塞。依然出现 502 错误

终极解决方法：

启用两个 php-fpm 实例 ,把 php-fpm 分为两部分 ,每部分各听一个端口或 socket ,这样就减少了 lock，

依然保持 400 个 php-fpm 进程，每个实例启用 200 个，采用 nginx 的 upstream 负载均衡，轮询每个

socket 来处理请求。

具体操作：

```
cp php-fpm.conf php-fpm2.conf
vi php-fpm2.conf 做相应的修改
[global]
pid = /usr/local/php/var/run/php-fpm2.pid
error_log = /usr/local/php/var/log/php-fpm2.log
log_level = notice
[www]
listen = /tmp/php-cgi2.sock
#listen = 127.0.0.1:9000
listen.backlog = 2048
listen.allowed_clients = 127.0.0.1
pm = static
pm.max_children = 200
pm.start_servers = 50
pm.min_spare_servers = 30
pm.max_spare_servers = 250
request_terminate_timeout = 0
request_slowlog_timeout = 2
slowlog = var/log/slow.log
cp /etc/init.d/php-fpm /etc/init.d/php-fpm2
vi  /etc/init.d/php-fpm2
修改
prefix=/usr/local/php
exec_prefix=${prefix}
php_fpm_BIN=${exec_prefix}/sbin/php-fpm
php_fpm_CONF=${prefix}/etc/php-fpm2.conf
php_fpm_PID=${prefix}/var/run/php-fpm2.pid
```

启动 php-fpm2 即可

配置 nginx

编辑 nginx.conf 主配置文件，如果后端采用虚拟主机，跟我一样，

添加

```
upstream backend{
        server unix:/tmp/php-cgi.sock;
        server unix:/tmp/php-cgi2.sock;
         }
```

vi vhost/test.conf

修改此处 fastcgi_pass backend; 调用 fastcgi 是，使用负载均衡的方式。

```
location ~ [^/]\.php(/|$)
                {
                        try_files $uri =404;
                        fastcgi_pass   backend;
                #        fastcgi_pass   127.0.0.1:9000;
                        fastcgi_index index.php;
                        include fastcgi.conf;
                       # include pathinfo.conf;
                }
```

重启 nginx。

等待验证吧，502 错误会大大地减少，网站抢购甚欢，消费者甚欢。

总结：

高并发时使用 tcp 端口的方式比 socket 方式相对稳定一点，但是使用端口的方式，处理的效率确实比 socket 效率低了那么一点。LNMP 环境下，在面对高并发时，除了一个合理的架构，与合理的调优之外，开发者的代码逻辑与高效的代码也是影响高并发的一个重要因素。一个请求调用多少次 php-fpm，每个 php-fpm 处理多少时间，都是开发者需要考虑的点。

安全运维之：网络性能评估工具 Iperf

作者：高俊峰 来源：http://ixdba.blog.51cto.com/2895551/1563110

一、 网络性能评估工具 Iperf

网络性能评估主要是监测网络带宽的使用率，将网络带宽利用最大化是保证网络性能的基础，但是由于网络设计不合理、网络存在安全漏洞等原因，都会导致网络带宽利用率不高。要找到网络带宽利用率不高的原因，就需要对网络传输进行监控，此时就需要用到一些网络性能评估工具，而 Iperf 就是这样一款网络带宽测试工具，本节将详细介绍一下 Iperf 的使用。

1、 Iperf 能做什么

Iperf 是一款基于 TCP/IP 和 UDP/IP 的网络性能测试工具，它可以用来测量网络带宽和网络质量，还可以提供网络延迟抖动、数据包丢失率、最大传输单元等统计信息。网络管理员可以根据这些信息了解并判断网络性能问题，从而定位网络瓶颈，解决网络故障。

下面介绍 Iperf 的主要功能。

（1）TCP 方面

q 测试网络带宽。

q 支持多线程，在客户端与服务端支持多重连接。

q 报告 MSS/MTU 值的大小。

q 支持 TCP 窗口值自定义并可通过套接字缓冲。

（2）UDP 方面

q 可以设置指定带宽的 UDP 数据流

q 可以测试网络抖动值、丢包数

q 支持多播测试

q 支持多线程，在客户端与服务端支持多重连接。

二、 Iperf 的安装与使用

iperf 可以运行在任何 IP 网络上，包括本地以太网、接入因特网、Wi-Fi 网络等。在工作模式上，iperf 运行于服务器、客户端模式下，其服务器端主要用于监听到达的测试请求，而客户端主要用于发起连接会话，因此要使用 iperf，需要两台服务器，一台运行在服务器模式下，另一台运行在客户端模式下。

1．安装 iperf

iperf 支持 Win32、Linux、FreeBSD、MacOS X、OpenBSD 和 Solaris 等多种操作系统平台。读者可以从 iperf 官方主页 http://iperf.fr/ 下载各种版本，目前最新的版本是 iperf3.0，这里下载的软件包为 iperf-3.0.tar.gz，安装过程如下：

[root@ networkserver ~]# tar zxvf iperf-3.0.tar.gz

[root@ networkserver ~]# cd iperf

[root@ networkserver iperf]# make

[root@ networkserver iperf]# make install

这样，iperf 就安装完成了。

2．iperf 参数介绍

在完成 iperf 安装后，执行“iperf3 –h”即可显示 iperf 的详细用法。iperf 的命令行选项共分为三类，分别是客户端与服务器端公用选项、服务器端专用选项和客户端专用选项，下面对常用的选项进行介绍。服务器端专用选项的含义如表 1 所示。

表 1 服务器端专用选项的含义

命令行参数	含义描述
-s	将 iperf 以 server 模式启动，例如：iperf3 –s，iperf3 默认启动的监听端口为 5201，可以通过“-p”选项修改默认监听端口
-D	将 iperf 作为后台守护进程运行，例如：iperf3 -s -D

客户端专用选项的含义如表 2.5 所示。

表 2 客户端专用选项的含义

命令行参数	含义描述
-c	将 iperf 以 client 模式启动 例如：iperf3 -c 192.168.12.168，其中 192.168.12.168 是 server 端的 IP 地址
-u	指定使用 UDP 协议
-b [K|M|G]	指定 UDP 模式使用的带宽，单位 bits/sec。此选项与“-u”选项相关。默认值是 1 Mbit/sec
-t	指定传输数据包的总时间。iperf 将在指定的时间内，重复发送指定长度的数据包。默认是 10 秒钟
-n [K|M|G]	指定传输数据包的字节数，例如：iperf3 -c 192.168.12.168 –n 100M
-l	指定读写缓冲区的长度。TCP 方式默认大小为 8KB，UDP 方式默认大小为 1470 字节
-P	指定客户端与服务端之间使用的线程数。默认是 1 个线程。需要客户端与服务器端同时使用此参数
-R	切换数据发送接收模式，例如默认客户端发送，服务器端接收，设置此参数后，数据流向变为客户端接收，服务器端发送
-w	指定套接字缓冲区大小，在 TCP 方式下，此设置为 TCP 窗口的大小。在 UDP 方式下，此设置为接受 UDP 数据包的缓冲区大小，用来限制可以接收数据包的最大值

-B	用来绑定一个主机地址或接口，这个参数仅用于具有多个网络接口的主机。在 UDP 模式下，此参数用于绑定和加入一个多播组
-M	设置 TCP 最大信息段的值
-N	设置 TCP 无延时

客户端与服务器端公用选项的含义如表 3 所示。

表 3 客户端与服务器端公用选项的含义

命令行参数	含义描述
-f[k|m|g|K|M|G]	指定带宽输出单位，“[k|m|g|K|M|G]”分别表示以 Kbits，Mbits，Gbits，KBytes，MBytes,GBytes 显示输出结果 默认以 Mbits 为单位 ,例如 :iperf3 -c 192.168.12.168 -f M
-p	指定服务器端使用的端口或客户端所连接的端口，例如： iperf3 -s -p 9527; iperf3 -c 192.168.12.168 -p 9527
-i	指定每次报告之间的时间间隔，单位为秒。如果设置为非零值，就会按照此时间间隔输出测试报告。默认值为 1。 例如：iperf3 -c 192.168.12.168 -i 2
-F	指定文件作为数据流进行带宽测试。 例如：iperf3 -c 192.168.12.168 -F web-ixdba.tar.gz

2.3.3 Iperf 应用实例

要使用 iperf，首先要启用一个服务端，这里假定服务端的 IP 地址为 192.168.12.168，在此服务器上运行“iperf3 -s”即可开启 iperf 的服务器模式。在默认情况下，iperf3 将在服务端打开一个 5201 监听端口，

此时就可以将另一台服务器作为客户端执行 iperf 功能测试了。

1．测试 TCP 吞吐量

为了确定网卡的最大吞吐量，可以在任意客户端运行 iperf 命令，iperf 将尝试从客户端尽可能快地向服务端发送数据请求，并且会输出发送的数据量和网卡平均带宽值。图 1 是一个最简单的带宽测试命令。

```
[root@networkserver app]# iperf3 -c 192.168.12.168
Connecting to host 192.168.12.168, port 5201
[  4] local 192.168.12.123 port 51628 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-1.00   sec   111 MBytes   929 Mbits/sec    3
[  4]   1.00-2.00   sec   113 MBytes   944 Mbits/sec    0
[  4]   2.00-3.00   sec   112 MBytes   944 Mbits/sec    0
[  4]   3.00-4.00   sec   112 MBytes   941 Mbits/sec    0
[  4]   4.00-5.00   sec   112 MBytes   944 Mbits/sec    0
[  4]   5.00-6.00   sec   112 MBytes   943 Mbits/sec    0
[  4]   6.00-7.00   sec   112 MBytes   943 Mbits/sec    0
[  4]   7.00-8.00   sec   112 MBytes   944 Mbits/sec    8
[  4]   8.00-9.00   sec   112 MBytes   942 Mbits/sec    0
[  4]   9.00-10.00  sec   112 MBytes   937 Mbits/sec    0
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-10.00  sec  1.10 GBytes   941 Mbits/sec   11          sender
[  4]   0.00-10.00  sec  1.09 GBytes   940 Mbits/sec               receiver

iperf Done.
```

图 1 通过 iperf 测试网络带宽利用率

从图 1 可以看出，iperf 默认的运行时间是 10 秒钟，每隔一秒钟输出一次传输状态，同时还可以看到每秒钟传输的数据量在 112MB 左右，刚好与“Bandwidth”列的值对应起来，网卡的带宽速率维持在 941Mbits/sec 左右，而测试的服务器是千兆网卡，这个测试值也基本合理。在输出的最后，iperf 还给出了总的数据发送、接收量，并给出了带宽速率平均值，通过这些值，基本可以判断网络带宽是否正常，网络传输状态是否稳定。

iperf 提供很多参数，可以多角度、全方位地测试网络带宽利用率，例如，要改变 iperf 运行的时间和输出频率，可以通过“-t”和“-i”参数来实现，如图 2 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -t 20 -i 5
Connecting to host 192.168.12.168, port 5201
[  4] local 192.168.12.123 port 55771 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-5.00   sec   560 MBytes   940 Mbits/sec   31
[  4]   5.00-10.00  sec   562 MBytes   942 Mbits/sec   18
[  4]  10.00-15.00  sec   561 MBytes   941 Mbits/sec    5
[  4]  15.00-20.00  sec   561 MBytes   942 Mbits/sec   19
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-20.00  sec  2.19 GBytes   941 Mbits/sec   73          sender
[  4]   0.00-20.00  sec  2.19 GBytes   941 Mbits/sec               receiver

iperf Done.
```

图 2 添加“-t”和“-i”参数后的 iperf 输出

从图 2 可以看出，输出状态的间隔变为每 5 秒钟一次，总共执行测试时间为 20 秒，测试的带宽速率仍然保持在 941Mbits/sec 左右，唯一变化的是失败重传次数增加了。

为了模拟大量的数据传输，也可以指定要发送的数据量，这可以通过“-n”参数来实现。在指定“-n”参数后，“-t”参数失效，iperf 在传输完毕指定大小的数据包后，自动结束，如图 3 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -i 10 -n 5000000000
Connecting to host 192.168.12.168, port 5201
[  4] local 192.168.12.123 port 58611 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-10.00  sec  1.09 GBytes   941 Mbits/sec  103
[  4]  10.00-20.00  sec  1.10 GBytes   942 Mbits/sec   45
[  4]  20.00-30.01  sec  1.10 GBytes   941 Mbits/sec   39
[  4]  30.01-40.00  sec  1.06 GBytes   907 Mbits/sec  484
[  4]  40.00-42.86  sec   321 MBytes   942 Mbits/sec   38
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-42.86  sec  4.66 GBytes   933 Mbits/sec  709          sender
[  4]   0.00-42.86  sec  4.66 GBytes   933 Mbits/sec               receiver

iperf Done.
```

图 3 iperf 客户端通过“-n”参数指定要传输的数据量

图 3 的例子是指定发送一个 5GB 左右的数据包，并且每隔 10 秒钟输出一次传输状态，从这个输出可以看出，当失败重传次数较多时，传输速率急速下降。

有时候，为了模拟更真实的 TCP 应用，iperf 客户端允许从一个特定的文件发送数据，这可以通过“-F”参数实现，如图 4 所示。

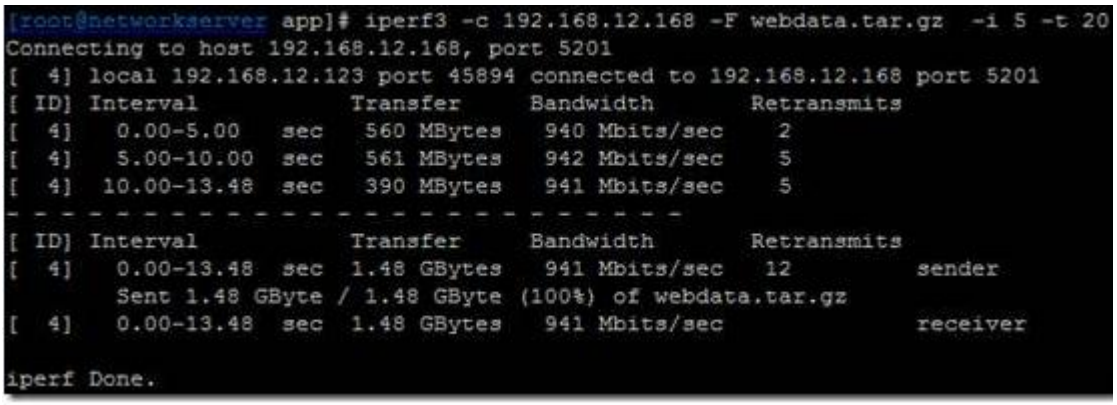


图 4 iperf 客户端通过“-F”参数指定文件来发送数据

在图 4 的例子中，通过“-F”参数指定了一个 webdata.tar.gz 文件作为 iperf 要传输的数据，在使用此参数时，需要同时指定一个“-t”参数来设置要测试传输的时间，这个时间尽量设置长一些，因为在默认传输时间 10 秒内，这个文件可能还没有传完。

在使用 iperf 进行网络带宽测试时，如果没有指定发送方式，iperf 客户端只会使用一个单一的线程，而 iperf 是支持多线程的，可以使用 iperf 提供的“-P”参数来设置多线程的数目，通过使用多线程，可以在一定程度上增加网络的吞吐量。

下面通过两个例子进行简单对比，图 5 是 iperf 使用单线程传输 1.86GBytes 数据所消耗的时间和带宽使用情况。为了速率单位统一，这里使用“-f”参数将输出结果都通过 MBytes 来显示。

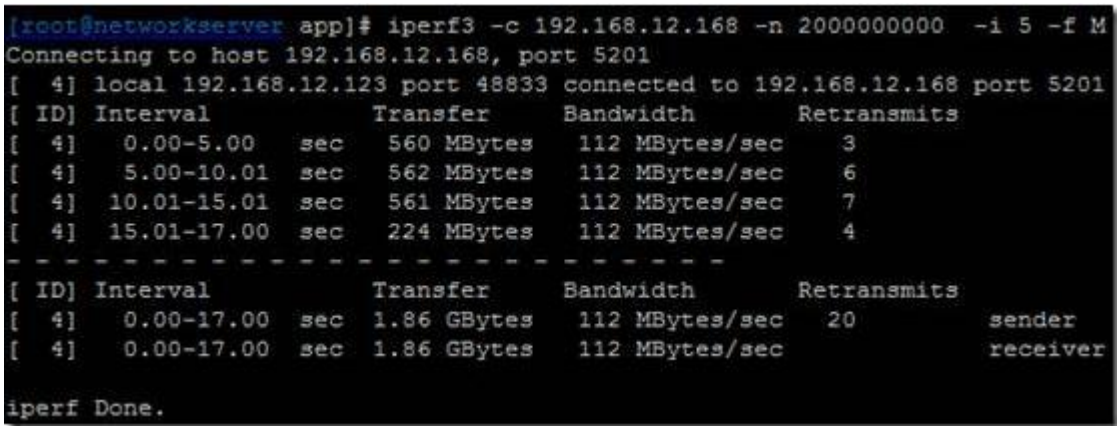


图 5 iperf 在单线程模式下的传输时间和传输速率

从图 5 中可以看出，传输 1.86GBytes 的数据消耗了 17 秒的时间，平均带宽速率为 112MBytes/sec（注意单位）.下面再看看使用多线程后，iperf 传输同样大小数据量所消耗的时间和平均带宽速率，如图 6 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -n 2000000000  -i 5 -P 2 -f M
Connecting to host 192.168.12.168, port 5201
[  4] local 192.168.12.123 port 48909 connected to 192.168.12.168 port 5201
[  6] local 192.168.12.123 port 48910 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-5.00   sec   430 MBytes  86.0 MBytes/sec  1618
[  6]   0.00-5.00   sec   430 MBytes  86.0 MBytes/sec  1756
[SUM]   0.00-5.00   sec   861 MBytes   172 MBytes/sec  3374
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   5.00-10.00  sec   446 MBytes  89.2 MBytes/sec  1925
[  6]   5.00-10.00  sec   446 MBytes  89.2 MBytes/sec  1856
[SUM]   5.00-10.00  sec   892 MBytes   178 MBytes/sec  3781
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]  10.00-10.79  sec  77.2 MBytes  98.3 MBytes/sec  582
[  6]  10.00-10.79  sec  77.2 MBytes  98.3 MBytes/sec  556
[SUM]  10.00-10.79  sec   154 MBytes   197 MBytes/sec  1138
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retransmits
[  4]   0.00-10.79  sec   954 MBytes  88.4 MBytes/sec  4125             sender
[  4]   0.00-10.79  sec   954 MBytes  88.4 MBytes/sec                   receiver
[  6]   0.00-10.79  sec   954 MBytes  88.4 MBytes/sec  4168             sender
[  6]   0.00-10.79  sec   954 MBytes  88.4 MBytes/sec                   receiver
[SUM]   0.00-10.79  sec  1.86 GBytes   177 MBytes/sec  8293             sender
[SUM]   0.00-10.79  sec  1.86 GBytes   177 MBytes/sec                   receiver

iperf Done.
```

图 6 iperf 使用多线程后的数据传输状态

这里通过“-P”参数开启了 2 个多线程，从传输时间上看，传输 1.86GBytes 的数据，消耗时间为 10.79 秒，比之前单线程的传输时间少了近 7 秒钟，在平均带宽速率上，从之前单线程的 112MBytes/sec 提高到 177MBytes/sec，从这个结果可以看出，多线程对网络传输性能的提高不小。

2．测试 UDP 丢包和延迟

iperf 也可以用于 UDP 数据包吞吐量、丢包率和延迟指标，但是由于 UDP 协议是一个非面向连接的轻量级传输协议，并且不提供可靠的数据传输服务，因此对 UDP 应用的关注点不是传输数据有多快，而是它的丢包率和延时指标。通过 iperf 的“-u”参数即可测试 UDP 应用的传输性能，图 7 测试的是在 iperf 客户端传输 100MB 的 UDP 数据包的输出结果.：

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -u -b 100M -f M -i 3
Connecting to host 192.168.12.168, port 5201
[  4] local 192.168.12.123 port 45516 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Total Datagrams
[  4]   0.00-3.00   sec  36.3 MBytes  12.1 MBytes/sec  4650
[  4]   3.00-6.00   sec  37.5 MBytes  12.5 MBytes/sec  4800
[  4]   6.00-9.00   sec  37.5 MBytes  12.5 MBytes/sec  4800
[  4]   9.00-10.00  sec  12.5 MBytes  12.5 MBytes/sec  1600
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  4]   0.00-10.00  sec   124 MBytes  12.4 MBytes/sec  0.052 ms  53/15850 (0.33%)
[  4] Sent 15850 datagrams

iperf Done.
```

图 7 iperf 传输 100MB 的 UDP 数据包的输出结果

在图 7 中，重点关注虚线下的一段内容，在这段输出中，“Jitter” 列表示抖动时间，或者称为传输延迟，“Lost/Total” 列表示丢失的数据报和总的数据报数量，后面的 0.33%是平均丢包的比率，“Datagrams” 列显示的是总共传输数据报的数量。

这个输出结果过于简单，要了解更详细的 UDP 丢包和延时信息，可以在 iperf 服务端查看，因为在客户端执行传输测试的同时，服务端也会同时显示传输状态，如图 8 所示。

```
[root@server app]# iperf3 -s -i 3
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.12.123, port 45108
[  5] local 192.168.12.168 port 5201 connected to 192.168.12.123 port 45516
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-3.00   sec  36.2 MBytes   101 Mbits/sec  0.053 ms  21/4650 (0.45%)
[  5]   3.00-6.00   sec  37.5 MBytes   105 Mbits/sec  0.047 ms  0/4800 (0%)
[  5]   6.00-9.00   sec  37.4 MBytes   105 Mbits/sec  0.047 ms  9/4800 (0.19%)
[  5]   9.00-10.04  sec  12.3 MBytes  99.4 Mbits/sec  0.052 ms  23/1600 (1.4%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  5]   0.00-10.04  sec   124 MBytes   103 Mbits/sec  0.052 ms  53/15850 (0.33%)
```

图 8 iperf 服务端显示的 UDP 传输状态

在这个输出中，详细记录了在传输过程中，每个阶段的传输延时和丢包率，在 UDP 应用中随着传输数据的增大，丢包率和延时也随之增加。对于延时和丢包可以通过改变应用程序来缓解或修复，例如视频流应用，可以通过缓存数据的方式而可以容忍更大的延时。

在 Puppet 中用 ERB 模板来自动配置 Nginx 虚拟主机

作者：余洪春 来源：http://yuhongchun.blog.51cto.com/1604432/1569791

模板文件是在 puppet 模块下面 templates 目录中以".erb"结尾的文件，puppet 模板主要用于文件，例如各种服务的配置文件，相同的服务，不同的配置就可以考虑使用模板文件，例如 Nginx 和 Apache 的虚拟主机配置就可以考虑采用 ERB 模板，nginx 的安装在这里建议用系统内部自带的 YUM 源来安装或其它第三方 YUM 源来安装，如果是用 Nginx 的官方源来安装 nginx 的话，我们可以查看下 /etc/yum.repos.d/nginx.repo 文件内容，如下所示：

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=0
enabled=1
```

第二种方式就是通过 createrepo 自建自己的 YUM 源，这种方式更加宁活，我们可以在 nginx 官网去下载适合自己的 rpm 包，然后添加进自己的 YUM 源，在自动化运维要求严格的定制环境中，绝大多数运维同学都会选择这种方法。大家通过此种方式安装 nginx 以后会发现，确实比源码安装 Nginx 方便多了，像自动分配了运行 nginx 的用户 nginx:nginx,Nginx 的日志存放会自动保存在/var/log/nginx 下，其工作目录为/etc/nginx，这一点跟源码编译安装的 nginx 区别很大，请大家在实验过程也注意甄别。

像 Puppet 其它初级知识点我这里就略过了，我直接贴上文件内容，/etc/puppet 的文件结构如下：

```
|-- auth.conf
|-- fileserver.conf
|-- manifests
|   |-- nodes
|   |   |-- client.cn7788.com.pp
|   |   `-- test.cn7788.com.pp
|   `-- site.pp
|-- modules
|   `-- nginx
|       |-- files
|       |-- manifests
|       |   `-- init.pp
|       `-- templates
|           |-- nginx.conf.erb
|           `-- nginx_vhost.conf.erb
`-- puppet.conf
```

site.pp 的文件内容如下：

```
1 import "nodes/*.pp"
```

client.cn7788.com.pp 的文件内容如下所示：

```
node 'client.cn7788.com' {
include nginx
  nginx::vhost {'client.cn7788.com':
  sitedomain => "client.cn7788.com" ,
  rootdir => "client",
}
}
```

test.cn7788.com.pp 的文件内容如下所示：

```
node 'test.cn7788.com' {
include nginx
nginx::vhost {'test.cn7788.com':
sitedomain => "test.cn7788.com" ,
rootdir => "test",
}
}
```

/etc/puppet/modules/nginx/manifests/init.pp 文件内容如下所示：

```
class nginx{
    package{"nginx":
    ensure          =>present,
}
    service{"nginx":
    ensure          =>running,
    require         =>Package["nginx"],
}
file{"nginx.conf":
ensure => present,
mode => 644,owner => root,group => root,
path => "/etc/nginx/nginx.conf",
content=> template("nginx/nginx.conf.erb"),
require=> Package["nginx"],
}
}
define nginx::vhost($sitedomain,$rootdir) {
    file{ "/etc/nginx/conf.d/${sitedomain}.conf":
    content => template("nginx/nginx_vhost.conf.erb"),
    require => Package["nginx"],
}
}
```

/etc/puppet/modules/nginx/templates/nginx.conf.erb 文件内容如下所示：

```
user  nginx;
worker_processes  8;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
events {
use epoll;
worker_connections  51200;
}
http {
include       /etc/nginx/mime.types;
default_type  application/octet-stream;
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';
access_log  /var/log/nginx/access.log  main;
sendfile        on;
#tcp_nopush     on;
keepalive_timeout  65;
#gzip  on;
include /etc/nginx/conf.d/*.conf;
}

}
```

/etc/puppet/modules/nginx/templates/nginx_vhost.conf.erb 文件内容如下所示：

```
server {
listen       80;
server_name  <%= sitedomain %>;
access_log /var/log/nginx/<%= sitedomain %>.access.log;
location / {
root /var/www/<%= rootdir %>;
index    index.php index.html index.htm;
}
}
```

最后我们可以在节点名为 client.cn7788.com 和 test.cn7788.com 的机器验证效果，命令如下所示：

```
1 puppetd --test --server server.cn7788.com
```

Android 笔记:触摸事件的分析与总结----多点触控

作者：glblong 来源：http://glblong.blog.51cto.com/3058613/1569058

一、多点触控

当多点同时触摸屏幕时，系统将会产生如下的触摸事件：

1.ACTION_DOWN：触摸屏幕的第一个点。此时手势开始。该点的数据通常在 MotionEvent 事件队列索引位置 0 处。

2.ACTION_POINTER_DOWN：除了第一个点的其他触摸点数据。该点的数据的索引位置由 getActionIndex（ ）方法返回。

3.ACTION_MOVE：在手势过程中发生的一次变化。

4.ACTION_POINTER_UP：当不是第一个点的其他点 UP 后触发。

5.ACTION_UP：当手势中的最后一个点离开屏幕。

简单测试范例

布局 xml 代码如下:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/layout1"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:tag="true 布局"
tools:context=".MainActivity" >

<TextView
    android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:tag="true 控件"
    android:text="@string/hello_world" />

</RelativeLayout>
```

java 代码如下:

```
package com.lonshine.d_touchduodian;

import android.os.Bundle;
```

```
import android.app.Activity;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.RelativeLayout;

public class MainActivity extends Activity implements OnTouchListener
{

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    RelativeLayout layout = (RelativeLayout) findViewById(R.id.layout1);
    layout.setOnTouchListener(this);

}

@Override
public boolean onTouch(View v, MotionEvent event)
{
    String tag = v.getTag().toString();
    Log.e(tag, "*********************************************");
    Log.e(tag, "*********************************************");
    Log.e(tag, "触摸的是:" + tag);
    Log.e(tag, describeEvent(event));
    logAction(event);
    Log.e(tag, "*********************************************");
    Log.e(tag, "*********************************************");
    Log.e("", "                      ");
    Log.e("", "                      ");

    if("true".equals(tag.substring(0, 4)))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
protected static String describeEvent(MotionEvent event)
{
    StringBuilder sb = new StringBuilder(500);

    sb.append("Action: ").append(event.getAction()).append("\n");// 获取触控动作比如 ACTION_DOWN
    int count = event.getPointerCount();
    sb.append("触点个数: ").append(count).append("\n");
    int i = 0;
    while (i < count)
    {
        sb.append("-------------------------").append("\n");
        int pointerId = event.getPointerId(i);
        sb.append("触点序号: ").append(i).append("\n");
        sb.append("触点 ID : ").append(pointerId).append("\n");
        sb.append("相对坐标: ").append(event.getX(i)).append("  *
").append(event.getY(i)).append("\n");
        sb.append("压力       : ").append(event.getPressure(i)).append("\n");// 压力值,0-1 之间,看情
况,很可能始终返回 1
        sb.append("范围大小: ").append(event.getSize(i)).append("\n");// 指压范围

        i++;
        sb.append("-------------------------").append("\n");
    }


    sb.append("按下时间: ").append(event.getDownTime()).append("ms   ");
    sb.append("结束时间: ").append(event.getEventTime()).append("ms   ");
    sb.append("运行时间: ").append(event.getEventTime() - event.getDownTime()).append("ms\n");

    return sb.toString();
}



private void logAction(MotionEvent event)
{
    int masked = event.getActionMasked();
    int index = event.getActionIndex();
    int pointerId = event.getPointerId(index);

    if(masked == 5 || masked == 6)
    {
        masked = masked - 5;
    }

    Log.e("Action", "触点序号: " + index);
```

```
    Log.e("Action", "触点 ID :   " + pointerId);
    Log.e("Action", "ActionMasked :" + masked);
}


}
```

多点触控的日志打印出来太多，此处仅针对日志输出作简单分析：

A.按下第一根手指时，获得一个索引为 0 且指针 ID 为 0 的指针(ACTION_DOWN = 0);

B.接下去 action 输出为 2(ACTION_MOVE)，此时仍然仅有一个指针，并且索引和 ID 都为 0;

C.然后按下第二根手指，action 值输出为 261。此值由两部分组成：一个是指针索引，一个是指针正在执行何种操作;

D.将十进制 261 转换为十六进制数为 0x00000105。其中 01 代表指针索引，05 代表操作值(即 ACTION_POINTER_DOWN 的值，用于多点触摸场景);

E.依此类推，当按下第三根手指时 action 输出值为 0x00000205(十进制 517)，第四根则为 0x000305(十进制 773);

F.当第一根手指离开屏幕时，action 值为 0x00000006(十进制 6)，即索引为 0，操作值为 6(即 ACTION_POINTER_UP);

G.如果此时第二根手指离开时,action 值应该为 0x00000106(十进制 262)，因为此时仍然拥有两根手指的信息;

H.而实际结果并没有得到 262 的 action 值，这是因为当第一根手指离开屏幕后，第二根手指的索引从 1 更改成了 0，但是指针 ID 仍然为 1.

I.对于每一次移动，action 值将始终为 2，因为 ACTION_MOVE 事件并没有包含哪根手指在移动的信息。

附：依次按下四根手指的日志

Application	Tag	Text
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	触摸的是:true布局
com.lonshine.d_touchduo...	true布局	Action: 0
com.lonshine.d_touchduo...	Action	ActionMasked :0
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...		
com.lonshine.d_touchduo...		
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	触摸的是:true布局
com.lonshine.d_touchduo...	true布局	Action: 261
com.lonshine.d_touchduo...	Action	ActionMasked :0
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...		
com.lonshine.d_touchduo...		
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	触摸的是:true布局
com.lonshine.d_touchduo...	true布局	Action: 517
com.lonshine.d_touchduo...	Action	ActionMasked :0
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...		
com.lonshine.d_touchduo...		
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	触摸的是:true布局
com.lonshine.d_touchduo...	true布局	Action: 773
com.lonshine.d_touchduo...	Action	ActionMasked :0
com.lonshine.d_touchduo...	true布局	**********
com.lonshine.d_touchduo...	true布局	**********

其他小结:

(1)getPointerCount() 获得触屏的点数，

getActionIndex()获得触点的指针索引，

getPointerId(index)获得指针索引对应的触点 ID；

(2)getActionMasked()表示用于多点触控检测点。而在 1.6 和 2.1 中并没有 event.getActionMasked()

这个方法，其实他就是把 event.getAction()& MotionEvent.ACTION_MASK 封装了一下。

二、其他参考范例：

(一)单点触摸拖动图片与多点触摸缩放图片

```
package com.lonshine.touchdemo;



import android.app.Activity;
import android.content.Context;
import android.graphics.*;
```

```
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.view.MotionEvent;
import android.widget.ImageView;
import android.widget.ImageView.ScaleType;

/**
 * 多点触控来控制 ImageView 中图像的放大与缩小，单点触控则用来    拖动图片
 *
 * @author zeng
 *
 */
public class MainActivity extends Activity
{
private MyImageView imageView;
private Bitmap bitmap;

// 两点触屏后之间的长度
private float beforeLenght;
private float afterLenght;

// 单点移动的前后坐标值
private float afterX, afterY;
private float beforeX, beforeY;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    findView();
    setContentView(imageView);
    config();
}

private void findView()
{
    imageView = new MyImageView(this);
    // 获得图片
    bitmap = ((BitmapDrawable) getResources().getDrawable(R.drawable.xing)).getBitmap();
}

private void config()
{
    // 设置 imageView 的显示图片
    imageView.setImageBitmap(bitmap);
```

```
    // 设置图片填充 ImageView
    imageView.setScaleType(ScaleType.FIT_XY);
}

// 创建一个自己的 ImageView 类
class MyImageView extends ImageView
{

    private float scale = 0.1f;

    public MyImageView(Context context)
    {
        super(context);
    }

    // 用来设置 ImageView 的位置
    private void setLocation(int x, int y)
    {
        this.setFrame(this.getLeft() + x, this.getTop() + y, this.getRight() + x, this.getBottom() +
y);
    }

    /**
         * 用来放大缩小 ImageView 因为图片是填充 ImageView 的，所以也就有放大缩小图片的效果 flag 为 0 是放大
图片，为 1 是缩小图片
     */
    private void setScale(float temp, int flag)
    {

        if (flag == 0)
        {
            this.setFrame(this.getLeft() - (int) (temp * this.getWidth()), this.getTop() - (int) (temp
* this.getHeight()), this.getRight()
                    + (int) (temp * this.getWidth()),   this.getBottom() + (int) (temp
* this.getHeight()));
        }
        else
        {
            this.setFrame(this.getLeft() + (int) (temp * this.getWidth()), this.getTop() + (int) (temp
* this.getHeight()), this.getRight()
                    - (int) (temp * this.getWidth()), this.getBottom() - (int) (temp *
this.getHeight()));
        }
    }

    // 绘制边框
```

```
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        Rect rec = canvas.getClipBounds();
        rec.bottom--;
        rec.right--;
        Paint paint = new Paint();
        paint.setColor(Color.RED);
        paint.setStyle(Paint.Style.STROKE);
        canvas.drawRect(rec, paint);
    }

    /**
     * 让图片跟随手指触屏的位置移动 beforeX、Y 是用来保存前一位置的坐标 afterX、Y 是用来保存当前位置的坐标
     * 它们的差值就是 ImageView 各坐标的增加或减少值
     */
    public void moveWithFinger(MotionEvent event)
    {

        switch (event.getAction())
        {

            case MotionEvent.ACTION_DOWN:
                beforeX = event.getX();
                beforeY = event.getY();
                break;
            case MotionEvent.ACTION_MOVE:
                afterX = event.getX();
                afterY = event.getY();

                this.setLocation((int) (afterX - beforeX), (int) (afterY - beforeY));

                beforeX = afterX;
                beforeY = afterY;
                break;

            case MotionEvent.ACTION_UP:
                break;
        }
    }

    /**
     * 通过多点触屏放大或缩小图像 beforeLenght 用来保存前一时间两点之间的距离 afterLenght 用来保存当前时间两点之间的距离
     */
```

```
    public void scaleWithFinger(MotionEvent event)
    {
        float moveX = event.getX(1) - event.getX(0);
        float moveY = event.getY(1) - event.getY(0);

        switch (event.getAction())
        {
            case MotionEvent.ACTION_DOWN:
                beforeLenght = (float) Math.sqrt((moveX * moveX) + (moveY * moveY));
                break;
            case MotionEvent.ACTION_MOVE:
                // 得到两个点之间的长度
                afterLenght = (float) Math.sqrt((moveX * moveX) + (moveY * moveY));

                float gapLenght = afterLenght - beforeLenght;

                if (gapLenght == 0)
                {
                    break;
                }

                // 如果当前时间两点距离大于前一时间两点距离，则传 0，否则传 1
                if (gapLenght > 0)
                {
                    this.setScale(scale, 0);
                }
                else
                {
                    this.setScale(scale, 1);
                }

                beforeLenght = afterLenght;
                break;
        }
    }

}

// 这里来监听屏幕触控时间
@Override
public boolean onTouchEvent(MotionEvent event)
{

    /**
         * 判定用户是否触摸到了图片 如果是单点触摸则调用控制图片移动的方法 如果是 2 点触控则调用控制图片大小
的方法
```

```
     */
    if (event.getY() > imageView.getTop() && event.getY() < imageView.getBottom() && event.getX() >
imageView.getLeft()
            && event.getX() < imageView.getRight())
    {
        if (event.getPointerCount() == 2)
        {
            imageView.scaleWithFinger(event);
        }
        else if (event.getPointerCount() == 1)
        {
            imageView.moveWithFinger(event);
        }
    }
    return true;
}
```

}

```
范例参考自：http://blog.csdn.net/ldj299/article/details/6422547
package com.lonshine.d_touchdemo;

import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.os.Handler;
import android.os.Message;
import android.view.MotionEvent;
import android.view.VelocityTracker;
import android.view.ViewConfiguration;


/**
 * 实现多点消息拦截，用于多个物体拖动等效果
 * @author zeng
 */
public class MultiTouchGestureDetector
{
@SuppressWarnings("unused")
private static final String MYTAG = "Alan";
public static final String CLASS_NAME = "MultiTouchGestureDetector";

/**
     * 事件信息类 <br/>
     * 用来记录一个手势
 */
```

```
private class EventInfo
{
    private MultiMotionEvent mCurrentDownEvent; // 当前的 down 事件
    private MultiMotionEvent mPreviousUpEvent; // 上一次 up 事件
    private boolean mStillDown; // 当前手指是否还在屏幕上
    private boolean mInLongPress; // 当前事件是否属于长按手势
    private boolean mAlwaysInTapRegion; // 是否当前手指仅在小范围内移动，当手指仅在小范围内移动时，视为手指未曾移动过，不会触发 onScroll 手势
    private boolean mAlwaysInBiggerTapRegion; // 是否当前手指在较大范围内移动，仅当此值为 true 时，双击手势才能成立
    private boolean mIsDoubleTapping; // 当前手势，是否为双击手势
    private float mLastMotionY; // 最后一次事件的 X 坐标
    private float mLastMotionX; // 最后一次事件的 Y 坐标

    private EventInfo(MotionEvent e)
    {
        this(new MultiMotionEvent(e));
    }

    private EventInfo(MultiMotionEvent me)
    {
        mCurrentDownEvent = me;
        mStillDown = true;
        mInLongPress = false;
        mAlwaysInTapRegion = true;
        mAlwaysInBiggerTapRegion = true;
        mIsDoubleTapping = false;
    }

    // 释放 MotionEven 对象，使系统能够继续使用它们
    public void recycle()
    {
        if (mCurrentDownEvent != null)
        {
            mCurrentDownEvent.recycle();
            mCurrentDownEvent = null;
        }
        if (mPreviousUpEvent != null)
        {
            mPreviousUpEvent.recycle();
            mPreviousUpEvent = null;
        }
    }

    @Override
    public void finalize()
```

```
    {
        this.recycle();
    }
}

/**
     * 多点事件类 <br/>
     * 将一个多点事件拆分为多个单点事件，并方便获得事件的绝对坐标 <br/>
     * 绝对坐标用以在界面中找到触点所在的控件
     *
     * @author ray-ni
 */
public class MultiMotionEvent
{
    private MotionEvent mEvent;
    private int mIndex;

    private MultiMotionEvent(MotionEvent e)
    {
        mEvent = e;
        mIndex = (e.getAction() & MotionEvent.ACTION_POINTER_ID_MASK) >>
MotionEvent.ACTION_POINTER_ID_SHIFT; // 等效于

// mEvent.getActionIndex();
    }

    private MultiMotionEvent(MotionEvent e, int idx)
    {
        mEvent = e;
        mIndex = idx;
    }

    // 行为
    public int getAction()
    {
        int action = mEvent.getAction() & MotionEvent.ACTION_MASK; // 等效于
                                                                   // mEvent.getActionMasked();
        switch (action)
        {
            case MotionEvent.ACTION_POINTER_DOWN:
                action = MotionEvent.ACTION_DOWN;
                break;
            case MotionEvent.ACTION_POINTER_UP:
                action = MotionEvent.ACTION_UP;
                break;
        }
```

```
        return action;
    }

    // 返回 X 的绝对坐标
    public float getX()
    {
        return mEvent.getX(mIndex) + mEvent.getRawX() - mEvent.getX();
    }

    // 返回 Y 的绝对坐标
    public float getY()
    {
        return mEvent.getY(mIndex) + mEvent.getRawY() - mEvent.getY();
    }

    // 事件发生的时间
    public long getEventTime()
    {
        return mEvent.getEventTime();
    }

    // 事件序号
    public int getIndex()
    {
        return mIndex;
    }

    // 事件 ID
    public int getId()
    {
        return mEvent.getPointerId(mIndex);
    }

    // 释放事件对象，使系统能够继续使用
    public void recycle()
    {
        if (mEvent != null)
        {
            mEvent.recycle();
            mEvent = null;
        }
    }
}

// 多点手势监听器
public interface MultiTouchGestureListener
```

```
{
    // 手指触碰到屏幕，由一个 ACTION_DOWN 触发
    boolean onDown(MultiMotionEvent e);

    // 确定一个 press 事件，强调手指按下的一段时间（TAP_TIMEOUT）内，手指未曾移动或抬起
    void onShowPress(MultiMotionEvent e);

    // 手指点击屏幕后离开，由 ACTION_UP 引发，可以简单的理解为单击事件，即手指点击时间不长（未构成长按事件），
也不曾移动过
    boolean onSingleTapUp(MultiMotionEvent e);

    // 长按，手指点下后一段时间（DOUBLE_TAP_TIMEOUT）内，不曾抬起或移动
    void onLongPress(MultiMotionEvent e);

    // 拖动，由 ACTION_MOVE 触发，手指地按下后，在屏幕上移动
    boolean onScroll(MultiMotionEvent e1, MultiMotionEvent e2, float distanceX, float distanceY);

    // 滑动，由 ACTION_UP 触发，手指按下并移动一段距离后，抬起时触发
    boolean onFling(MultiMotionEvent e1, MultiMotionEvent e2, float velocityX, float velocityY);
}

// 多点双击监听器
public interface MultiTouchDoubleTapListener
{
    // 单击事件确认，强调第一个单击事件发生后，一段时间内，未发生第二次单击事件，即确定不会触发双击事件
    boolean onSingleTapConfirmed(MultiMotionEvent e);

    // 双击事件，
    // 由 ACTION_DOWN 触发，从第一次单击事件的 DOWN 事件开始的一段时间(DOUBLE_TAP_TIMEOUT)内结束（即手指），
    // 并且在第一次单击事件的 UP 时间开始后的一段时间内（DOUBLE_TAP_TIMEOUT）发生第二次单击事件，
    // 除此之外两者坐标间距小于定值（DOUBLE_TAP_SLAP）时，则触发双击事件
    boolean onDoubleTap(MultiMotionEvent e);

    // 双击事件，与 onDoubleTap 事件不同之处在于，构成双击的第二次点击的 ACTION_DOWN,ACTION_MOVE 和
ACTION_UP 都会触发该事件
    boolean onDoubleTapEvent(MultiMotionEvent e);
}

// 事件信息队列，队列的下标与 MotionEvent 的 pointId 对应
private static List<EventInfo> sEventInfos = new ArrayList<EventInfo>(10);
// 双击判断队列，这个队列中的元素等待双击超时的判断结果
private static List<EventInfo> sEventForDoubleTap = new ArrayList<EventInfo>(5);
// 指定大点击区域的大小（这个比较拗口），这个值主要用于帮助判断双击是否成立
private int mBiggerTouchSlopSquare = 20 * 20;
// 判断是否构成 onScroll 手势，当手指在这个范围内移动时，不触发 onScroll 手势
private int mTouchSlopSquare;
```

```
// 判断是否构成双击，只有两次点击的距离小于该值，才能构成双击手势
private int mDoubleTapSlopSquare;
// 最小滑动速度
private int mMinimumFlingVelocity;
// 最大滑动速度
private int mMaximumFlingVelocity;

// 长按阀值，当手指按下后，在该阀值的时间内，未移动超过 mTouchSlopSquare 的距离并未抬起，则长按手势触发
private static final int LONGPRESS_TIMEOUT = ViewConfiguration.getLongPressTimeout();
// showPress 手势的触发阀值，当手指按下后，在该阀值的时间内，未移动超过 mTouchSlopSquare 的距离并未抬起，
则 showPress 手势触发
private static final int TAP_TIMEOUT = ViewConfiguration.getTapTimeout();
// 双击超时阀值，仅在两次双击事件的间隔（第一次单击的 UP 事件和第二次单击的 DOWN 事件）小于此阀值，双击事件才
能成立
private static final int DOUBLE_TAP_TIMEOUT = ViewConfiguration.getDoubleTapTimeout();
// 双击区域阀值，仅在两次双击事件的距离小于此阀值，双击事件才能成立
private static final int DOUBLE_TAP_SLAP = 64;

// GestureHandler 所处理的 Message 的 what 属性可能为以下 常量：
// showPress 手势
private static final int SHOW_PRESS = 1;
// 长按手势
private static final int LONG_PRESS = 2;
// SingleTapConfirmed 手势
private static final int TAP_SINGLE = 3;
// 双击手势
private static final int TAP_DOUBLE = 4;

// 手势处理器
private final GestureHandler mHandler;
// 手势监听器
private final MultiTouchGestureListener mListener;
// 双击监听器
private MultiTouchDoubleTapListener mDoubleTapListener;

// 长按允许阀值
private boolean mIsLongpressEnabled;
// 速度追踪器
private VelocityTracker mVelocityTracker;

private class GestureHandler extends Handler
{
    GestureHandler()
    {
        super();
    }
```

```
GestureHandler(Handler handler)
{
    super(handler.getLooper());
}

@Override
public void handleMessage(Message msg)
{
    int idx = (Integer) msg.obj;
    switch (msg.what)
    {
        case SHOW_PRESS:
        {
            if (idx >= sEventInfos.size())
            {
                // Log.w(MYTAG, CLASS_NAME +
                // ":handleMessage, msg.what = SHOW_PRESS, idx=" + idx +
                // ", while sEventInfos.size()="
                // + sEventInfos.size());
                break;
            }
            EventInfo info = sEventInfos.get(idx);
            if (info == null)
            {
                // Log.e(MYTAG, CLASS_NAME +
                // ":handleMessage, msg.what = SHOW_PRESS, idx=" + idx +
                // ", Info = null");
                break;
            }
            // 触发手势监听器的 onShowPress 事件
            mListener.onShowPress(info.mCurrentDownEvent);
            break;
        }
        case LONG_PRESS:
        {
            // Log.d(MYTAG, CLASS_NAME + ":trigger LONG_PRESS");

            if (idx >= sEventInfos.size())
            {
                // Log.w(MYTAG, CLASS_NAME +
                // ":handleMessage, msg.what = LONG_PRESS, idx=" + idx +
                // ", while sEventInfos.size()="
                // + sEventInfos.size());
                break;
            }
```

```
                EventInfo info = sEventInfos.get(idx);
                if (info == null)
                {
                    // Log.e(MYTAG, CLASS_NAME +
                    // ":handleMessage, msg.what = LONG_PRESS, idx=" + idx +
                    // ", Info = null");
                    break;
                }
                dispatchLongPress(info, idx);
                break;
            }
            case TAP_SINGLE:
            {
                // Log.d(MYTAG, CLASS_NAME + ":trriger TAP_SINGLE");
                // If the user's finger is still down, do not count it as a
                // tap
                if (idx >= sEventInfos.size())
                {
                    // Log.e(MYTAG, CLASS_NAME +
                    // ":handleMessage, msg.what = TAP_SINGLE, idx=" + idx +
                    // ", while sEventInfos.size()="
                    // + sEventInfos.size());
                    break;
                }
                EventInfo info = sEventInfos.get(idx);
                if (info == null)
                {
                    // Log.e(MYTAG, CLASS_NAME +
                    // ":handleMessage, msg.what = TAP_SINGLE, idx=" + idx +
                    // ", Info = null");
                    break;
                }
                if (mDoubleTapListener != null && !info.mStillDown)
                { // 手指在双击超时的阀值内未离开屏幕进行第二次单击事件，则确定单击事件成立（不再触发双击事
件）
                    mDoubleTapListener.onSingleTapConfirmed(info.mCurrentDownEvent);
                }
                break;
            }
            case TAP_DOUBLE:
            {
                if (idx >= sEventForDoubleTap.size())
                {
                    // Log.w(MYTAG, CLASS_NAME +
                    // ":handleMessage, msg.what = TAP_DOUBLE, idx=" + idx +
                    // ", while sEventForDoubleTap.size()="
```

```
                    // + sEventForDoubleTap.size());
                    break;
                }
                EventInfo info = sEventForDoubleTap.get(idx);
                if (info == null)
                {
                    // Log.w(MYTAG, CLASS_NAME +
                    // ":handleMessage, msg.what = TAP_DOUBLE, idx=" + idx +
                    // ", Info = null");
                    break;
                }
                sEventForDoubleTap.set(idx, null);// 这个没什么好做的，就是把队列中对应的元素清除而已
                break;
            }
            default:
                throw new RuntimeException("Unknown message " + msg); // never
        }
    }
}

/**
     * 触发长按事件
     *
     * @param info
     * @param idx
 */
private void dispatchLongPress(EventInfo info, int idx)
{
    mHandler.removeMessages(TAP_SINGLE, idx);// 移除单击事件确认
    info.mInLongPress = true;
    mListener.onLongPress(info.mCurrentDownEvent);
}

/**
     * 构造器 1
     *
     * @param context
     * @param listener
 */
public MultiTouchGestureDetector(Context context, MultiTouchGestureListener listener)
{
    this(context, listener, null);
}

/**
     * 构造器 2
```

```
     *
     * @param context
     * @param listener
     * @param handler
 */
public MultiTouchGestureDetector(Context context, MultiTouchGestureListener listener, Handler
handler)
{
    if (handler != null)
    {
        mHandler = new GestureHandler(handler);
    }
    else
    {
        mHandler = new GestureHandler();
    }
    mListener = listener;
    if (listener instanceof MultiTouchDoubleTapListener)
    {
        setOnDoubleTapListener((MultiTouchDoubleTapListener) listener);
    }
    init(context);
}

/**
     * 初始化识别器
     *
     * @param context
 */
private void init(Context context)
{
    if (mListener == null)
    {
        throw new NullPointerException("OnGestureListener must not be null");
    }
    mIsLongpressEnabled = true;
    int touchSlop, doubleTapSlop;
    if (context == null)
    {
        touchSlop = ViewConfiguration.getTouchSlop();
        doubleTapSlop = DOUBLE_TAP_SLAP;
        mMinimumFlingVelocity = ViewConfiguration.getMinimumFlingVelocity();
        mMaximumFlingVelocity = ViewConfiguration.getMaximumFlingVelocity();
    }
    else
    {// 允许识别器在 App 中，使用偏好的设定
```

```
        final ViewConfiguration configuration = ViewConfiguration.get(context);
        touchSlop = configuration.getScaledTouchSlop();
        doubleTapSlop = configuration.getScaledDoubleTapSlop();
        mMinimumFlingVelocity = configuration.getScaledMinimumFlingVelocity();
        mMaximumFlingVelocity = configuration.getScaledMaximumFlingVelocity();
    }
    mTouchSlopSquare = touchSlop * touchSlop / 16;
    mDoubleTapSlopSquare = doubleTapSlop * doubleTapSlop;
}

/**
     * 设置双击监听器
     *
     * @param onDoubleTapListener
 */
public void setOnDoubleTapListener(MultiTouchDoubleTapListener onDoubleTapListener)
{
    mDoubleTapListener = onDoubleTapListener;
}

/**
     * 设置是否允许长按
     *
     * @param isLongpressEnabled
 */
public void setIsLongpressEnabled(boolean isLongpressEnabled)
{
    mIsLongpressEnabled = isLongpressEnabled;
}

/**
     * 判断是否允许长按
     *
     * @return
 */
public boolean isLongpressEnabled()
{
    return mIsLongpressEnabled;
}

/**
     * 判断当前事件是否为双击事件 <br/>
     * 通过遍历 sEventForDoubleTap 来匹配是否存在能够构成双击事件的单击事件
     *
     * @param e
     * @return
```

```
 */
private EventInfo checkForDoubleTap(MultiMotionEvent e)
{
    if (sEventForDoubleTap.isEmpty())
    {
        // Log.e(MYTAG, CLASS_NAME +
        // ":checkForDoubleTap(), sEventForDoubleTap is empty !");
        return null;
    }
    for (int i = 0; i < sEventForDoubleTap.size(); i++)
    {
        EventInfo info = sEventForDoubleTap.get(i);
        if (info != null && isConsideredDoubleTap(info, e))
        {
            sEventForDoubleTap.set(i, null);// 这个单击事件已经被消耗了，所以置为 null
            mHandler.removeMessages(TAP_DOUBLE, i);// 移除 Handler 内的为处理消息
            return info;
        }
    }
    return null;
}

/**
     * 判断当前按下事件是否能和指定的单击事件构成双击事件
     *
     * @param info
     * @param secondDown
     * @return
     */
private boolean isConsideredDoubleTap(EventInfo info, MultiMotionEvent secondDown)
{
    if (!info.mAlwaysInBiggerTapRegion)
    { // 如多第一次单击事件有过较大距离的移动，则无法构成双击事件
        return false;
    }
    if (secondDown.getEventTime() - info.mPreviousUpEvent.getEventTime() > DOUBLE_TAP_TIMEOUT)
    {
        // 如果第一次单击的 UP 时间和第二次单击的 down 时间时间间隔大于 DOUBLE_TAP_TIMEOUT，也无法构成双击
事件
        return false;
    }
    int deltaX = (int) info.mCurrentDownEvent.getX() - (int) secondDown.getX();
    int deltaY = (int) info.mCurrentDownEvent.getY() - (int) secondDown.getY();
    return (deltaX * deltaX + deltaY * deltaY < mDoubleTapSlopSquare);// 最后判断两次单击事件的距离
}
```

```
/**
     * 将事件信息放入双击判断队列，并返回序号
     *
     * @param info
     * @return
     */
private int addIntoTheMinIndex(EventInfo info)
{
    for (int i = 0; i < sEventForDoubleTap.size(); i++)
    {
        if (sEventForDoubleTap.get(i) == null)
        {
            sEventForDoubleTap.set(i, info);
            return i;
        }
    }
    sEventForDoubleTap.add(info);
    return sEventForDoubleTap.size() - 1;
}

/**
     * 从事件信息队列中移除指定序号的事件
     *
     * @param idx
 */
private void removeEventFromList(int id)
{
    if (id > sEventInfos.size() || id < 0)
    {
        // Log.e(MYTAG, CLASS_NAME + ".removeEventFromList(), id=" + id +
        // ", while sEventInfos.size() =" +
        // sEventInfos.size());
        return;
    }
    sEventInfos.set(id, null);
}

/**
     * 向事件队列中添加新信息
     *
     * @param e
 */
private void addEventIntoList(EventInfo info)
{
    int id = info.mCurrentDownEvent.getId();
    if (id < sEventInfos.size())
```

```
    {
        // if (sEventInfos.get(id) != null)
        // Log.e(MYTAG, CLASS_NAME + ".addEventIntoList, info(" + id +
        // ") has not set to null !");
        sEventInfos.set(info.mCurrentDownEvent.getId(), info);
    }
    else if (id == sEventInfos.size())
    {
        sEventInfos.add(info);
    }
    else
    {
        // Log.e(MYTAG, CLASS_NAME + ".addEventIntoList, invalidata id !");
    }
}

public boolean onTouchEvent(MotionEvent ev)
{
    if (mVelocityTracker == null)
    {
        mVelocityTracker = VelocityTracker.obtain();
    }
    mVelocityTracker.addMovement(ev);// 把所有事件都添加到速度追踪器，为计算速度做准备
    boolean handled = false;
    final int action = ev.getAction(); // 获取 Action
    // int idx = (action & MotionEvent.ACTION_POINTER_INDEX_MASK) >>
    // MotionEvent.ACTION_POINTER_INDEX_SHIFT;//获取触摸事件的序号
    int idx = ev.getPointerId(ev.getActionIndex());// 获取触摸事件的 id
    switch (action & MotionEvent.ACTION_MASK)
    {
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_POINTER_DOWN:
        {
            EventInfo info = new EventInfo(MotionEvent.obtain(ev));
            this.addEventIntoList(info);// 将手势信息保存到队列中
            if (mDoubleTapListener != null)
            {// 如果双击监听器不为 null
                if (mHandler.hasMessages(TAP_DOUBLE))
                {
                    MultiMotionEvent e = new MultiMotionEvent(ev);
                    EventInfo origInfo = checkForDoubleTap(e);// 检查是否构成双击事件
                    if (origInfo != null)
                    {
                        info.mIsDoubleTapping = true;
                        handled |= mDoubleTapListener.onDoubleTap(origInfo.mCurrentDownEvent);
                        handled |= mDoubleTapListener.onDoubleTapEvent(e);
```

```
                }
            }
            if (!info.mIsDoubleTapping)
            {// 当前事件不构成双击事件，那么发送延迟消息以判断 onSingleTapConfirmed 事件
                mHandler.sendMessageDelayed(mHandler.obtainMessage(TAP_SINGLE, idx),
DOUBLE_TAP_TIMEOUT);
                // Log.d(MYTAG, CLASS_NAME + ": add TAP_SINGLE");
            }
        }
        // 记录 X 坐标和 Y 坐标
        info.mLastMotionX = info.mCurrentDownEvent.getX();
        info.mLastMotionY = info.mCurrentDownEvent.getY();

        if (mIsLongpressEnabled)
        {// 允许长按
            mHandler.removeMessages(LONG_PRESS, idx);
            mHandler.sendMessageAtTime(mHandler.obtainMessage(LONG_PRESS, idx),
info.mCurrentDownEvent.getEventTime() + TAP_TIMEOUT + LONGPRESS_TIMEOUT);// 延时消息以触发长按手势
            // Log.d(MYTAG, CLASS_NAME +
            // ":add LONG_PRESS to handler  for idx " + idx);
        }
        mHandler.sendMessageAtTime(mHandler.obtainMessage(SHOW_PRESS, idx),
info.mCurrentDownEvent.getEventTime() + TAP_TIMEOUT);// 延时消息，触发 showPress 手势
        handled |= mListener.onDown(info.mCurrentDownEvent);// 触发 onDown（）
        break;
    }
    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
    {
        MultiMotionEvent currentUpEvent = new MultiMotionEvent(ev);
        if (idx >= sEventInfos.size())
        {
            // Log.e(MYTAG, CLASS_NAME + ":ACTION_POINTER_UP, idx=" +
            // idx + ", while sEventInfos.size()=" +
            // sEventInfos.size());
            break;
        }
        EventInfo info = sEventInfos.get(currentUpEvent.getId());
        if (info == null)
        {
            // Log.e(MYTAG, CLASS_NAME + ":ACTION_POINTER_UP, idx=" +
            // idx + ", Info = null");
            break;
        }
        info.mStillDown = false;
        if (info.mIsDoubleTapping)
```

```
            { // 处于双击状态，则触发 onDoubleTapEvent 事件
                handled |= mDoubleTapListener.onDoubleTapEvent(currentUpEvent);
            }
            else if (info.mInLongPress)
            {// 处于长按状态
                mHandler.removeMessages(TAP_SINGLE, idx);// 可以无视这行代码
                info.mInLongPress = false;
            }
            else if (info.mAlwaysInTapRegion)
            {// 尚未移动过
                if (mHandler.hasMessages(TAP_SINGLE, idx))
                {// 还在双击的时间阀值内，所以要为双击判断做额外处理
                    mHandler.removeMessages(TAP_SINGLE, idx);
                    info.mPreviousUpEvent = new MultiMotionEvent(MotionEvent.obtain(ev));
                    int index = this.addIntoTheMinIndex(info);// 把当前事件放入队列，等待双击的判断
                    mHandler.sendMessageAtTime(mHandler.obtainMessage(TAP_DOUBLE, index),
info.mCurrentDownEvent.getEventTime() + DOUBLE_TAP_TIMEOUT); // 将双击超时判断添加到 Handler
                    // Log.d(MYTAG, CLASS_NAME + ": add TAP_DOUBLE");
                }
                handled = mListener.onSingleTapUp(currentUpEvent);// 触发 onSingleTapUp 事件
            }
            else
            {
                // A fling must travel the minimum tap distance
                final VelocityTracker velocityTracker = mVelocityTracker;
                velocityTracker.computeCurrentVelocity(1000, mMaximumFlingVelocity);// 计算 1 秒钟内的
滑动速度
                // 获取 X 和 Y 方向的速度
                final float velocityX = velocityTracker.getXVelocity(idx);
                final float velocityY = velocityTracker.getYVelocity(idx);
                // Log.i(MYTAG, CLASS_NAME + ":ACTION_POINTER_UP, idx=" +
                // idx +
                // ", vx=" + velocityX + ", vy=" + velocityY);
                // 触发滑动事件
                if ((Math.abs(velocityY) > mMinimumFlingVelocity) || (Math.abs(velocityX) >
mMinimumFlingVelocity))
                {
                    handled = mListener.onFling(info.mCurrentDownEvent, currentUpEvent, velocityX,
velocityY);
                }
            }
            // Hold the event we obtained above - listeners may have changed
            // the
            // original.
            if (action == MotionEvent.ACTION_UP)
            { // 释放速度追踪器
```

```
                mVelocityTracker.recycle();
                mVelocityTracker = null;
                // Log.w(MYTAG, CLASS_NAME +
                // ":ACTION_POINTER_UP, mVelocityTracker.recycle()");
            }

            info.mIsDoubleTapping = false;
            // Log.d(MYTAG, CLASS_NAME + "remove LONG_PRESS");
            // 移除 showPress 和长按消息
            mHandler.removeMessages(SHOW_PRESS, idx);
            mHandler.removeMessages(LONG_PRESS, idx);
            removeEventFromList(currentUpEvent.getId());// 手指离开，则从队列中删除手势信息
            break;
        }
        case MotionEvent.ACTION_MOVE:
            for (int rIdx = 0; rIdx < ev.getPointerCount(); rIdx++)
            {// 因为无法确定当前发生移动的是哪个手指，所以遍历处理所有手指
                MultiMotionEvent e = new MultiMotionEvent(ev, rIdx);
                if (e.getId() >= sEventInfos.size())
                {
                    // Log.e(MYTAG, CLASS_NAME + ":ACTION_MOVE, idx=" + rIdx
                    // + ", while sEventInfos.size()=" +
                    // sEventInfos.size());
                    break;
                }
                EventInfo info = sEventInfos.get(e.getId());
                if (info == null)
                {
                    // Log.e(MYTAG, CLASS_NAME + ":ACTION_MOVE, idx=" + rIdx
                    // + ", Info = null");
                    break;
                }
                if (info.mInLongPress)
                { // 长按，则不处理 move 事件
                    break;
                }
                // 当前坐标
                float x = e.getX();
                float y = e.getY();
                // 距离上次事件移动的位置
                final float scrollX = x - info.mLastMotionX;
                final float scrollY = y - info.mLastMotionY;
                if (info.mIsDoubleTapping)
                {// 双击事件
                    handled |= mDoubleTapListener.onDoubleTapEvent(e);
                }
```

```
                else if (info.mAlwaysInTapRegion)
                {// 该手势尚未移动过（移动的距离小于 mTouchSlopSquare,视为未移动过）
                    // 计算从落下到当前事件，移动的距离
                    final int deltaX = (int) (x - info.mCurrentDownEvent.getX());
                    final int deltaY = (int) (y - info.mCurrentDownEvent.getY());
                    // Log.d(MYTAG, CLASS_NAME + "deltaX="+deltaX+";deltaY="
                    // +
                    // deltaX +"mTouchSlopSquare=" + mTouchSlopSquare);
                    int distance = (deltaX * deltaX) + (deltaY * deltaY);
                    if (distance > mTouchSlopSquare)
                    { // 移动距离超过 mTouchSlopSquare
                        handled = mListener.onScroll(info.mCurrentDownEvent, e, scrollX, scrollY);
                        info.mLastMotionX = e.getX();
                        info.mLastMotionY = e.getY();
                        info.mAlwaysInTapRegion = false;
                        // Log.d(MYTAG, CLASS_NAME +
                        // ":remove LONG_PRESS for idx" + rIdx +
                        // ",mTouchSlopSquare("+mTouchSlopSquare+"), distance("+distance+")");
                        // 清除 onSingleTapConform，showPress,longPress 三种消息
                        int id = e.getId();
                        mHandler.removeMessages(TAP_SINGLE, id);
                        mHandler.removeMessages(SHOW_PRESS, id);
                        mHandler.removeMessages(LONG_PRESS, id);
                    }
                    if (distance > mBiggerTouchSlopSquare)
                    {// 移动距离大于 mBiggerTouchSlopSquare，则无法构成双击事件
                        info.mAlwaysInBiggerTapRegion = false;
                    }
                }
                else if ((Math.abs(scrollX) >= 1) || (Math.abs(scrollY) >= 1))
                {// 之前已经移动过了
                    handled = mListener.onScroll(info.mCurrentDownEvent, e, scrollX, scrollY);
                    info.mLastMotionX = x;
                    info.mLastMotionY = y;
                }
            }
            break;
        case MotionEvent.ACTION_CANCEL:
            cancel();// 清理
    }
    return handled;
}

// 清理所有队列
private void cancel()
{
```

```
    mHandler.removeMessages(SHOW_PRESS);
    mHandler.removeMessages(LONG_PRESS);
    mHandler.removeMessages(TAP_SINGLE);
    mVelocityTracker.recycle();
    mVelocityTracker = null;
    sEventInfos.clear();
    sEventForDoubleTap.clear();
}
}
```

tomcat + memcached session manager 共享 session

作者：老徐_kevin 来源：http://laoxu.blog.51cto.com/4120547/1566477

网上有很多关于通过 MSM(memcached session manager)实现 memcached 共享 session 的文章，但是很多都是东拼西凑，误导别人。正巧最近有一个地方用到，特此总结一下。

MSM 支持 tomcat6，tomcat7，tomcat8，MSM 支持两种模式:sticky sessions（粘性 session）和 non-sticky sessions（非粘性 session）。我用到的是 sticky session，所以以下都按照 sticky session 来介绍。集群结构是 2 个 tomcat 实例节点，2 个 memcached 实例节点

<tomcat1> <tomcat2>

.\/.

machine1 . X . machine2

./\.

<memcached1> <memcached2>

简单说明下：tomcat1 为主要把它的 session 存储到 memcached2 上，而 memcache2 是运行在另一台主机上的（一般情况下，memcached 是存 tomcat1 的 session），只有当 memcached2 不可用时，tomcat1 才会把 session 存在 memcached1 上，也就是说 memcached1 是为了 tomcat1 做故障切换用的节点。这要的话，当 machine1 挂了的时候，session 是不会丢失的。

接着我们考虑使用 session 的哪种序列化方式，默认的是使用 java 进行序列化，是由 memcached-session-manager.jar 这个 jar 包来提供的方法，而其它的序列化方式是由其它的 jar 包提供的。

首先是要安装 jdk 和 tomcat，这里不再赘述，当然 tomcat 可以选择支持 native 函数。在修改 tomcat 配置文件之前，先把一些 jar 包放在 $CATALINA_HOME/lib/（WEB-INF/lib）目录里，再修改 $CATALINA_HOME/conf/ (META-INF/context.xml)配置文件。

我们使用的是 memcached，所以需要 spymemcached-2.11.1.jar

补充：如果使用 couchbase，需要添加以下 jar 包：couchbase-client-1.4.0.jar jettison-1.3.jar, commons-codec-1.5.jar, httpcore-4.3.jar,httpcore-nio-4.3.jar, netty-3.5.5.Final.jar

需要注意的是，如果你使用 java 内置的序列化方式，把 jar 放在$CATALINA_HOME/lib/里即可。如果为了更好的性能，使用自定义的序列化方式，就要把其它 jar 包部署在具体 java 项目工程下的 WEB-INF/lib 里。

以下是四种 session 序列化方式对应需要的 jar 包

kryo-serializer: msm-kryo-serializer, kryo-serializers-0.11 (0.11 is needed, as 0.20+ is for kryo2), kryo, minlog, reflectasm, asm-3.2

javolution-serializer: msm-javolution-serializer, javolution-5.4.3.1

xstream-serializer: msm-xstream-serializer, xstream, xmlpull, xpp3_min

flexjson-serializer: msm-flexjson-serializer, flexjson

所以非粘性 sessions 使用 kryo 序列化所需要增加的 jar 包如下：

```
asm-3.2.jar
kryo-1.04.jar
kryo-serializers-0.11.jar
memcached-session-manager-1.6.3.jar
memcached-session-manager-tc6-1.6.3.jar
minlog-1.2.jar
msm-kryo-serializer-1.6.3.jar
reflectasm-1.01.jar
spymemcached-2.11.1.jar
```

虽然我使用的是 kryo 序列化方式 + 非粘性 session，但是还是把粘性 session 一起介绍一下。

sticky sessions 粘性 session + kryo 序列化

在 machine1 上有 tomcat1 和 memcached1，tomcat 的$CATALINA_HOME/conf/context.xml 增加如下配置

```
<Context>
  ...
  <Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
    memcachedNodes="n1:host1.example.com:11211,n2:host2.example.com:11211"
    failoverNodes="n1"
    requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
    transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"
    />
</Context>
```

参数 failoverNodes="n1" 是用来告诉 MSM 把 session 优先存储在 memcached2 上，只有当 memcached2 挂了的时候才会把 session 存在 memcache1 也就是配置文件的 n1 里。

假如整个 machine1 挂了，session 还是可用，因为 session 是存在 machine2 的 memcache2 上，可以通过 tomcat2 对外提供服务。

machine2 上的 tomcat2 的配置文件只要改成 failoverNodes="n2"即可。

non-sticky sessions 非粘性 session + kryo 序列化

非粘性 session 是不需要配置 failoverNodes，也就是故障切换节点的，因此 session 是由所有 tomcat 节点通过轮训（round-robin ）来提供服务的，session 是不和某单个 tomcat 节点绑定，所以 tomcat 所有节点的都是一样的，如下：

```
<Context>
...
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="n1:host1.example.com:11211,n2:host2.example.com:11211"
sticky="false"
sessionBackupAsync="false"
lockingMode="uriPattern:/path1|/path2"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"
```

```
/>
</Context>
```

配置完成后重新启动两个节点上的不同 tomcat 服务。

```
1 tail -f /usr/local/tomcat/logs/catalina.out
```

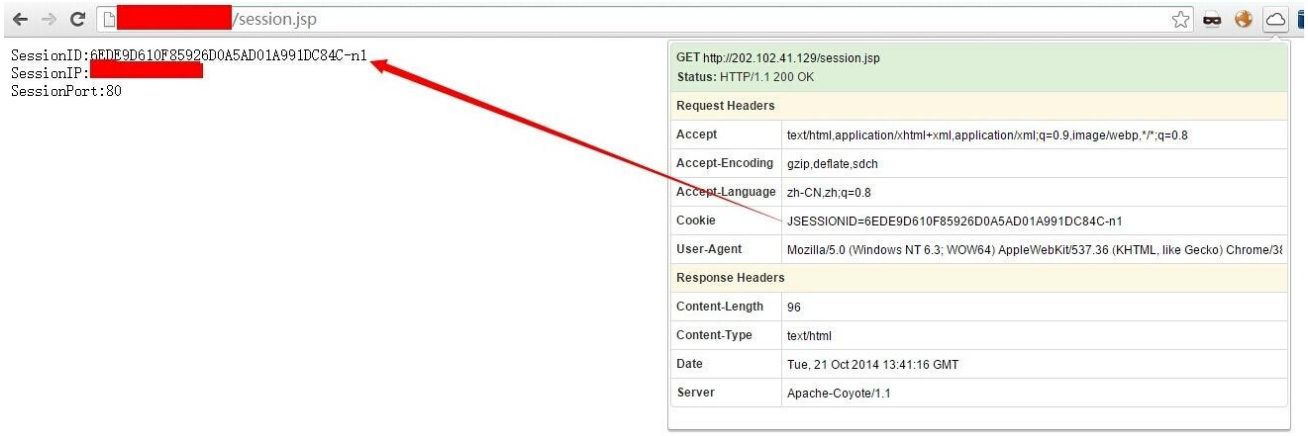
```
2014-10-21 21:25:25.251 INFO net.spy.memcached.MemcachedConnection:  Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService setLockingMode
INFO: Setting lockingMode to URI_PATTERN with pattern /path1|/path2
Connection state changed for sun.nio.ch.SelectionKeyImpl@172fb0af
2014-10-21 21:25:25.252 INFO Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService createTranscoderFactory
INFO: Creating transcoder factory de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory
net.spy.memcached.MemcachedConnection:  Connection state changed for sun.nio.ch.SelectionKeyImpl@3bf8bd0d
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.serializer.kryo.KryoTranscoder <init>
INFO: Starting with initialBufferSize 102400 and maxBufferSize 2048000
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService startInternal
INFO: MemcachedSessionService finished initialization, sticky false, operation timeout 1000, with node ids [n1, n2] and failover node ids []
Oct 21, 2014 9:25:25 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory examples
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService startInternal
INFO: MemcachedSessionService starts initialization... (configured nodes definition n1:192.168.203.166:11211,n2:192.168.203.167:11211, failover nodes null)
2014-10-21 21:25:25.481 INFO net.spy.memcached.MemcachedConnection:  Added {QA sa=/192.168.203.166:11211, #Rops=0, #Wops=0, #iq=0, topRop=null, topWop=null, toWrite=0,
interested=0} to connect queue
2014-10-21 21:25:25.482 INFO net.spy.memcached.MemcachedConnection:  Added {QA sa=/192.168.203.167:11211, #Rops=0, #Wops=0, #iq=0, topRop=null, topWop=null, toWrite=0,
interested=0} to connect queue
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.RequestTrackingHostValve <init>
INFO: Setting ignorePattern to .*\.(ico|png|gif|jpg|css|js)$
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService setLockingMode
INFO: Setting lockingMode to URI_PATTERN with pattern /path1|/path2
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService createTranscoderFactory
INFO: Creating transcoder factory de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory
2014-10-21 21:25:25.487 Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.serializer.kryo.KryoTranscoder <init>
INFO: Starting with initialBufferSize 102400 and maxBufferSize 2048000
INFO net.spy.memcached.MemcachedConnection:  Connection state changed for sun.nio.ch.SelectionKeyImpl@ec0a9f9
2014-10-21 21:25:25.488 INFO net.spy.memcached.MemcachedConnection:  Connection state changed for sun.nio.ch.SelectionKeyImpl@163b4b1e
Oct 21, 2014 9:25:25 PM de.javakaffee.web.msm.MemcachedSessionService startInternal
INFO: MemcachedSessionService finished initialization, sticky false, operation timeout 1000, with node ids [n1, n2] and failover node ids []
Oct 21, 2014 9:25:25 PM org.apache.coyote.http11.Http11AprProtocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Oct 21, 2014 9:25:25 PM org.apache.coyote.ajp.AjpAprProtocol start
INFO: Starting Coyote AJP/1.3 on ajp-8009
Oct 21, 2014 9:25:25 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1601 ms
```

两个节点 tomcat 的启动日志都正常。

测试的 session.jsp 页面内容如下

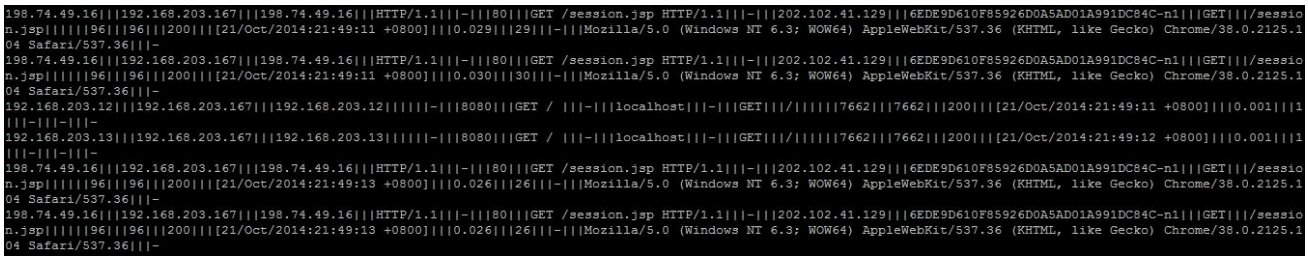
```
SessionID:<%=session.getId()%> <BR>
SessionIP:<%=request.getServerName()%> <BR>
SessionPort:<%=request.getServerPort()%>
```

浏览器访问测试页面



页面获取的和我 chrome 浏览器截获的 cookie 里 session id 是一样的，这里补充下 session id 是保存在 cookie 里的。

多次刷新页面 session.jsp，发现 session id 不变化，再查看访问 tomcat 的访问日志，出现多条访问记录（这里补充说明，tomcat 默认是关闭访问日志的，需要开启，然后自定义日志格式。）但是 session 保持不变。



由于 session 默认是存在 tomcat 的 jvm 里的 ,我们验证一下 memcached 里是不是真的存了这个 session 记录，查询 memcached 存的内容然后导出到文件里

```
1 | echo "stats cachedump 3 0" | nc 192.168.203.167 11211 > session.txt
```

```
1 | grep '6EDE9D610F85926D0A5AD01A991DC84C-n1 ' session.txt
```

确认存在。

至此 tomcat 的 MSM，共享 session 的工作就做完了，good luck！

给 HashMap 做个深度剖析手术

作者：NightWolves 来源：http://yangfei520.blog.51cto.com/1041581/1566942

人们对于任何事物的认知，往往都存在这么一个现象：只有你了解的东西，你才会感兴趣。

HashMap 之于 Java 开发者来说，也是如此。相信多数人在起初相当长的时间内，对 HashMap 的印象都是“Map 接口的实现类，是基于哈希的，用于存放键-值对，允许 null 作为键和值，非线程安全的”，仅此而已。于是在程序编写过程中便“肆无忌惮”往里放键-值对。而只有你对 HashMap 的实现有了一定的了解之后，你才会有兴趣研究 HashMap 深层次的问题，比如“HashMap 最多能放多少个键-值对？如何提高 HashMap 的使用效率？”。其实，我一直都对 HashMap 的“线性数组+链表”的实现机制充满好奇，刚刚有时间研究了一下源码，现把心得与大家分享。

首先，我们来看看 HashMap 的数据结构：

```
/**
 *  初始容量，大小必须是 2 的指数次方，默认是 16.
 */
static final int DEFAULT_INITIAL_CAPACITY = 16;
/**
 * 默认最大容量，值为 1<<30
 */
static final int MAXIMUM_CAPACITY = 1 << 30;
/**
 * 哈希表的默认加载因子，值为 0.75.
 */
static final float DEFAULT_LOAD_FACTOR = 0.75f;
/**
 * 存储元素的数组，大小必须是 2 的指数次方，默认是 16.
 */
transient Entry[] table;
/**
 * THashMap 中的存储的<K,V>映射的数目.
 */
transient int size;
/**
 * threshold=容量*加载因子。当实际数目大于 threshold 时，HashMap 就需要扩容.
 * @serial
 */
int threshold;
/**
```

```
 * 哈希表的加载因子，如果创建时不指定 loadFactor，则使用 DEFAULT_LOAD_FACTOR.
 *
 * @serial
 */
final float loadFactor;
//... ...
/**
 *
 * <p>用一个静态内部类来定义数组项链表的元素</p>
 *
 * @author bruce.yang
 * @version 1.0 Created on 2014-10-22 上午 11:39:50
 */
static class Entry<K,V> implements Map.Entry<K,V> {
    final K key;
    V value;
    Entry<K,V> next;
    final int hash;
    /**
     * Creates new entry.
     */
    Entry(int h, K k, V v, Entry<K,V> n) {
        value = v;
        next = n;
        key = k;
        hash = h;
    }
    public final K getKey() {
        return key;
    }
    public final V getValue() {
        return value;
    }
    public final V setValue(V newValue) {
 V oldValue = value;
        value = newValue;
        return oldValue;
    }
    public final boolean equals(Object o) {
        if (!(o instanceof Map.Entry))
            return false;
        Map.Entry e = (Map.Entry)o;
        Object k1 = getKey();
        Object k2 = e.getKey();
        if (k1 == k2 || (k1 != null && k1.equals(k2))) {
            Object v1 = getValue();
```

```
            Object v2 = e.getValue();
            if (v1 == v2 || (v1 != null && v1.equals(v2)))
                return true;
        }
        return false;
    }
    public final int hashCode() {
        return (key==null   ? 0 : key.hashCode()) ^
               (value==null ? 0 : value.hashCode());
    }
    public final String toString() {
        return getKey() + "=" + getValue();
    }
```

可以看出，HashMap 正是采用数组(table)类存储数据的，而数组每一个元素则是一个被静态内部类 Entry 封装起来的对象，元素在数组中的下标则是根据 key 的 hashcode 计算出来的；当元素下标重复的时候，会在此下标处形成一个链表，而这个链表就是通过 Entry 结构来实现的。我们看到，这个 Entry 结构是一个单向链表，它只有一个 next 项指向下一个元素，此外，它还包含<K,V>对，同时还有一个哈希值 hash。

其次，当我们往 hashmap 中 put 元素的时候，先是根据 key 的 hash 值计算得出这个元素在数组中的位置(即下标),然后就可以把这个元素放到对应的位置中了。如果这个位子上已经存放有其他元素了，那么在同一个位子上的元素将以链表的形式存放，新加入的放在链头，最先加入的放在链尾。源码：

```
/**
 * 建立指定 key 和 value 之间的映射：
 * 如果已经存在 key，则替换对应的 value，并返回原来的 value；如果原来没有建立映射，则建立映射，并返回 null。
 * 当然了，由于 HashMap 允许 key 和 value 为 null，返回 null 还有可能就是原来的 value 为 null。
 */
public V put(K key, V value) {
 //如果 key==null，则调用专门的方法处理
    if (key == null)
        return putForNullKey(value);
    int hash = hash(key.hashCode());
    int i = indexFor(hash, table.length);
    for (Entry<K,V> e = table[i]; e != null; e = e.next) {
        Object k;
        if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {
            V oldValue = e.value;
            e.value = value;
            e.recordAccess(this);
```

```
            //原来存在这项
            return oldValue;
        }
    }
    modCount++;
    //加入新的项
    addEntry(hash, key, value, i);
    //原来不存在这项
    return null;
}
private V putForNullKey(V value) {
    for (Entry<K,V> e = table[0]; e != null; e = e.next) {
        if (e.key == null) {
            V oldValue = e.value;
            e.value = value;
            e.recordAccess(this);
            return oldValue;
        }
    }
    modCount++;
    addEntry(0, null, value, 0);
    return null;
}
/**
 * 添加 key-value 项
 */
void addEntry(int hash, K key, V value, int bucketIndex) {
 Entry<K,V> e = table[bucketIndex];
 //加入该项到链表头部
    table[bucketIndex] = new Entry<K,V>(hash, key, value, e);
    if (size++ >= threshold)
        resize(2 * table.length);
}
/**
 *扩充 HashMap 的容量
 */
void resize(int newCapacity) {
    Entry[] oldTable = table;
    int oldCapacity = oldTable.length;
    if (oldCapacity == MAXIMUM_CAPACITY) {
        threshold = Integer.MAX_VALUE;
        return;
    }
    Entry[] newTable = new Entry[newCapacity];
    transfer(newTable);
    table = newTable;
```

```
    threshold = (int)(newCapacity * loadFactor);
}
/**
 * 将原来数组中元素传输到新数组中
 */
void transfer(Entry[] newTable) {
    Entry[] src = table;
    int newCapacity = newTable.length;
    for (int j = 0; j < src.length; j++) {
        Entry<K,V> e = src[j];
        if (e != null) {
            src[j] = null;
            do {
                Entry<K,V> next = e.next;
                int i = indexFor(e.hash, newCapacity);
                e.next = newTable[i];
                newTable[i] = e;
                e = next;
            } while (e != null);
        }
    }
}
```

可以看到,首先判断 key 是否为 null:

1).若为 null，则调用专门的方法 putForNullKey（value）处理并返回。

1.1)如果事先已经存在 key 为 null 的映射，则替换后返回 old value。

1.2)如果不存在，则添加新的项到链表中

2).若 key 不为 null

2.1)首先计算 key 的哈希值，然后根据哈希值和 table 数组的长度定位数组项。

2.2)对数组项的链表进行遍历，如果 key 的哈希值与链表中的某一项的哈希值相等且 key 本身引用值相等或者引用值所指向的对象相等，则替换相应项的 value 值为新的 value，并返回老的 value。如果没有找到相同的 key，则加入该项到链表中。

2.3)addEntry 方法直接将新的项加入到链表的头部，新项的 next 引用指向原来的链表项。此外判断是否需要扩容 如果此时存储的项数目 size 大于等于 threshold 则扩大 HashMap 容量为原来的 2 倍。

2.4)resize 方法用来扩容 HashMap。默认是扩容至原来的 2 倍大小。

第三，当从 hashmap 中 get 元素时，首先计算 key 的 hashcode，找到数组中对应位置的某一元素，然后通过 key 的 equals 方法在对应位置的链表中找到需要的元素。源码：

```
/**
 * 根据 key 取得 value
 */
public V get(Object key) {
    if (key == null)
        return getForNullKey();
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash, table.length)];
         e != null;
         e = e.next) {
        Object k;
        if (e.hash == hash && ((k = e.key) == key || key.equals(k)))
            return e.value;
    }
    return null;
}
private V getForNullKey() {
    for (Entry<K,V> e = table[0]; e != null; e = e.next) {
        if (e.key == null)
            return e.value;
    }
    return null;
}
```

可以看到，如果 key 为 null，get 方法与 put 方法对应，对 null 值也有特殊处理，即直接到 table[0] 中去找 key 为 null 对应的 value。

如果 key 不为 null，则定位 key 所在数组项，然后遍历链表，如果存在 key，则返回对应的 value 值，否则返回 null。

第四，HashMap 的初始化：

```
/**
 *指定初始化容量和加载因子
 */
public HashMap(int initialCapacity, float loadFactor) {
 //initialCapacity 不能小于 0
    if (initialCapacity < 0)
        throw new IllegalArgumentException("Illegal initial capacity: " +
                                           initialCapacity);
    //如果指定容量大于默认最大容量，则按照默认最大容量创建
```

```
    if (initialCapacity > MAXIMUM_CAPACITY)
        initialCapacity = MAXIMUM_CAPACITY;
    //loadFactor 不能小于 0
    if (loadFactor <= 0 || Float.isNaN(loadFactor))
        throw new IllegalArgumentException("Illegal load factor: " +
                                           loadFactor);
    //找最大于指定初始化容量的 2 的指数幂作为表的真实容量
    int capacity = 1;
    while (capacity < initialCapacity)
        capacity <<= 1;
    this.loadFactor = loadFactor;
    threshold = (int)(capacity * loadFactor);
    table = new Entry[capacity];
    init();
}
/**
 *只指定初始化容量，加载因子采用默认
 */
public HashMap(int initialCapacity) {
    this(initialCapacity, DEFAULT_LOAD_FACTOR);
}
/**
 * 初始化容量和加载因子都采用默认
 */
public HashMap() {
    this.loadFactor = DEFAULT_LOAD_FACTOR;
    threshold = (int)(DEFAULT_INITIAL_CAPACITY * DEFAULT_LOAD_FACTOR);
    table = new Entry[DEFAULT_INITIAL_CAPACITY];
    init();
}
```

只看源码和注释就一目了然了，这里所要特别注意一下带两个参数的构造方法，其初始化时的初始大小不一定是你所指定的大小。

最后，文章开头提出的那两个问题已经不再成为问题了：

1).HashMap 最大容量是 1 << 30，

2).而影响 HashMap 性能的两个因素是：初始容量和加载因子。这里借用书中的一段话来表述：容量是哈希表中桶的数量，初始容量只是哈希表在创建时的容量。加载因子是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 rehash 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。

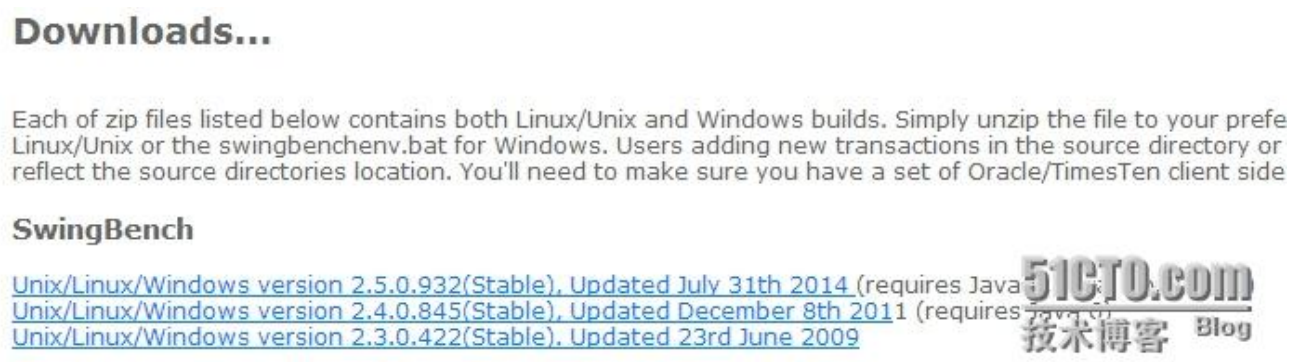
通常，默认加载因子 (0.75)在时间和空间成本上寻求一种折衷。加载因子过高虽然减少了空间开销，但同时也增加了查询成本（在大多数 HashMap 类的操作中，包括 get 和 put 操作，都反映了这一点）。在设置初始容量时应该考虑到映射中所需的条目数及其加载因子，以便最大限度地减少 rehash 操作次数。如果初始容量大于最大条目数除以加载因子，则不会发生 rehash 操作。

读到这里，相信你对 HashMap 已经有了一个更深刻的认识。

如何使用 swingbench 进行 oracle 数据库压力测试

作者：孙杰 来源：http://xjsunjie.blog.51cto.com/999372/1560779

swingbench 是一 款网络上开源的 oracle 压力测试工具，支持 oracle 11g 版本，还能对 rac 进行测试。从官方页面 http://dominicgiles.com/downloads.html 上可以下载最新的软件版本。



swingbench 可以运行在 windows 和 linux 平台，本次测试采用 linux 平台，具体测试过程如下：

1、首先使用 VMVARE10 搭建一个 redhat6.5 的虚拟机。

2、在虚拟 redhat6.5 上安装 ORACLE11G 的数据库。

3、使用 oewizard 导入测试数据，可以根据向导提示进行数据导入。

4、使用 swingbench 进行压力测试。

测试步骤：

1、导数据之前需要修改 temp 表空间大小，使其能够容纳下相应的导入数据

```
create temporary tablespace temp1 tempfile '/home/oracle/oradata/orcl/temp1.dbf' size 100m;

alter database default temporary tablespace temp1;

drop tablespace TEMP;

create temporary tablespace temp tempfile '/home/oracle/oradata/orcl/temp.dbf' size 1g;

alter database default temporary tablespace temp;
```

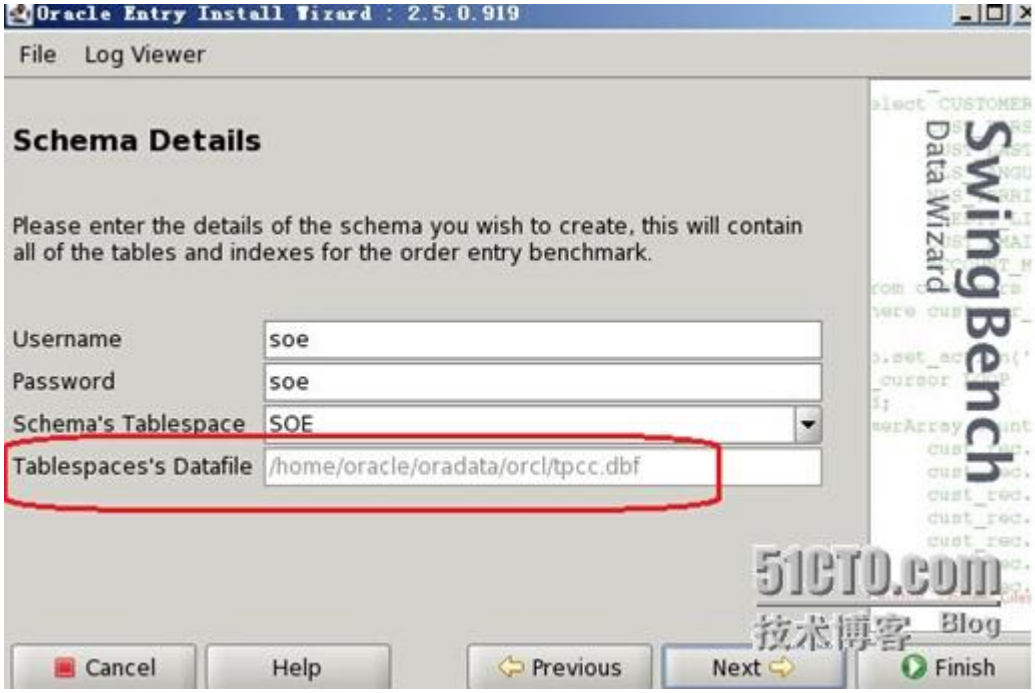
drop tablespace temp1;

2、安装 swingbench 测试软件，直接解压缩即可运行。

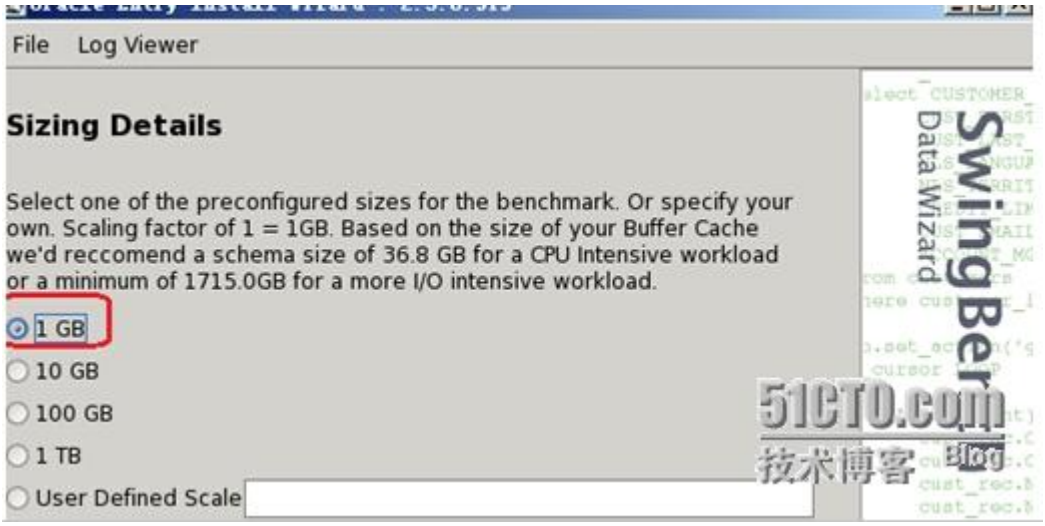
unzip -x swingbench25919.zip

3、进入 swingbench/bin 目录执行 oewizard 导入 1G 测试数据，并修改数据库连接名和 DBA 密码

输入导入数据文件存放位置：

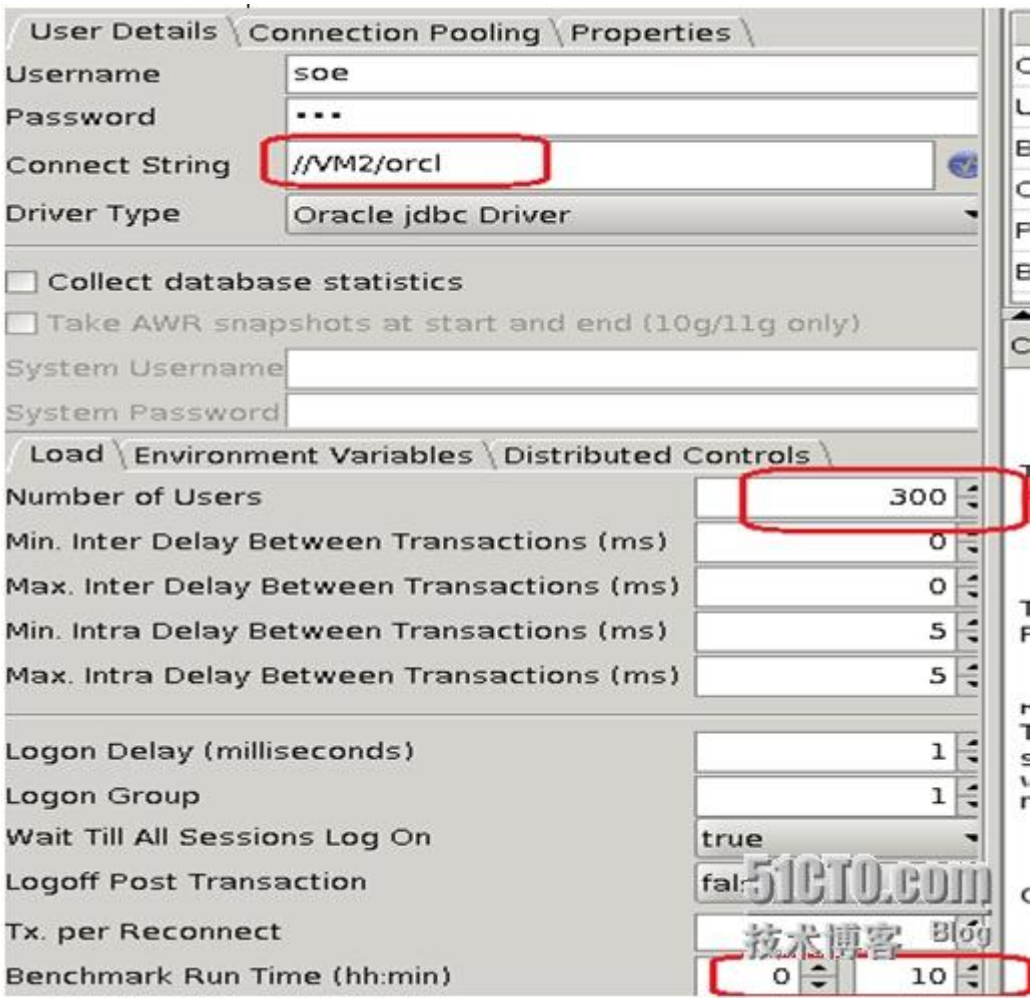


选择导入 1G 数据：

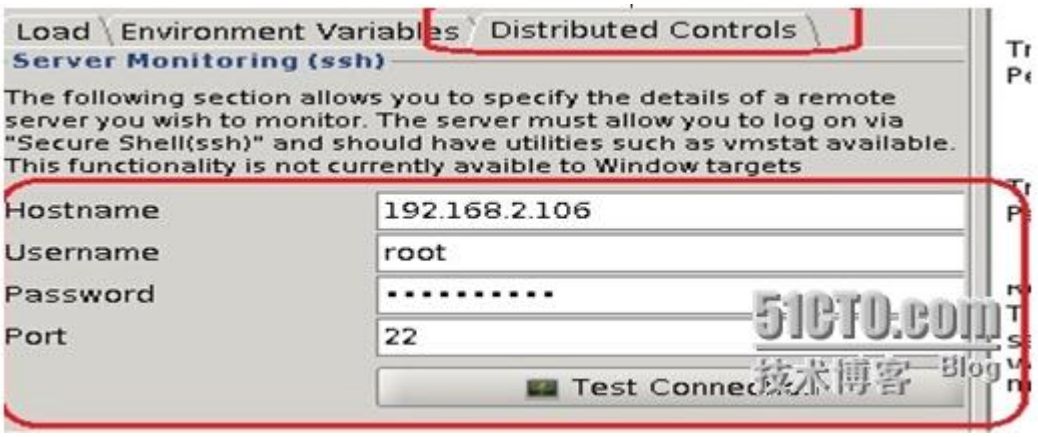


数据导完之后在该目录下运行 swingbench 执行测试，修改数据库连接名，用户连接数设置为 300，测试

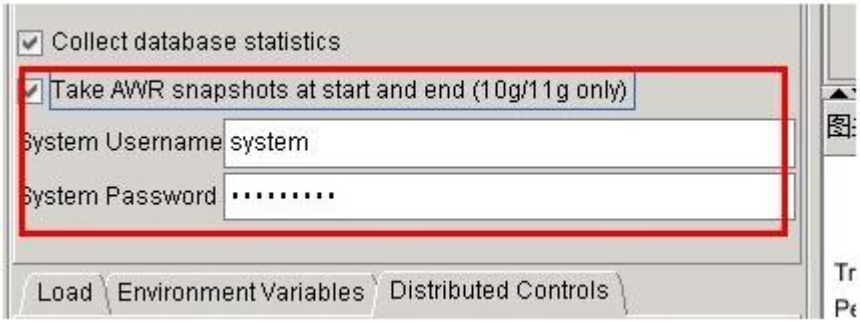
时间设置为 10 分钟



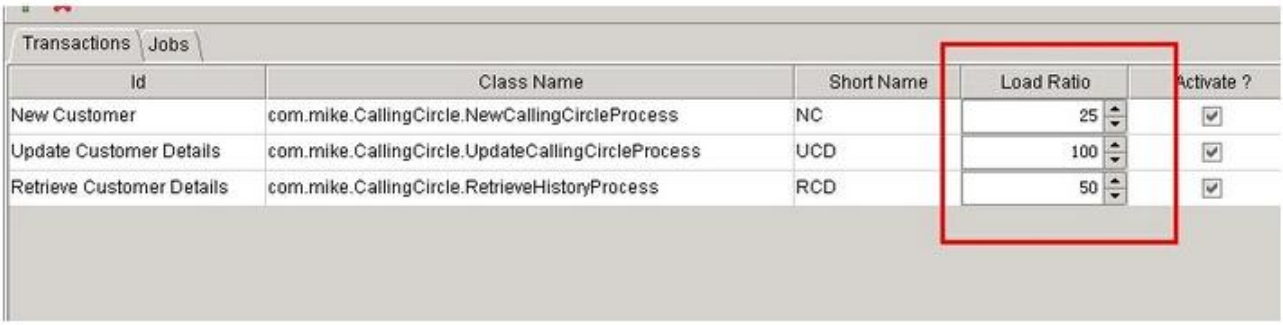
修改 Distributed Controls 用于测试过程中搜集测试监控信息，修改完之后测试连接是否正常，并可以统计主机的 cpu disk IO 信息



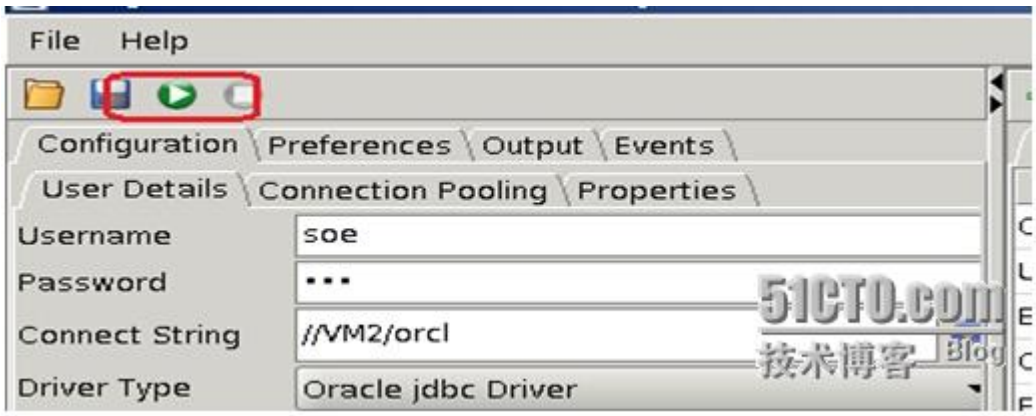
还可以拉出 AWR 报表



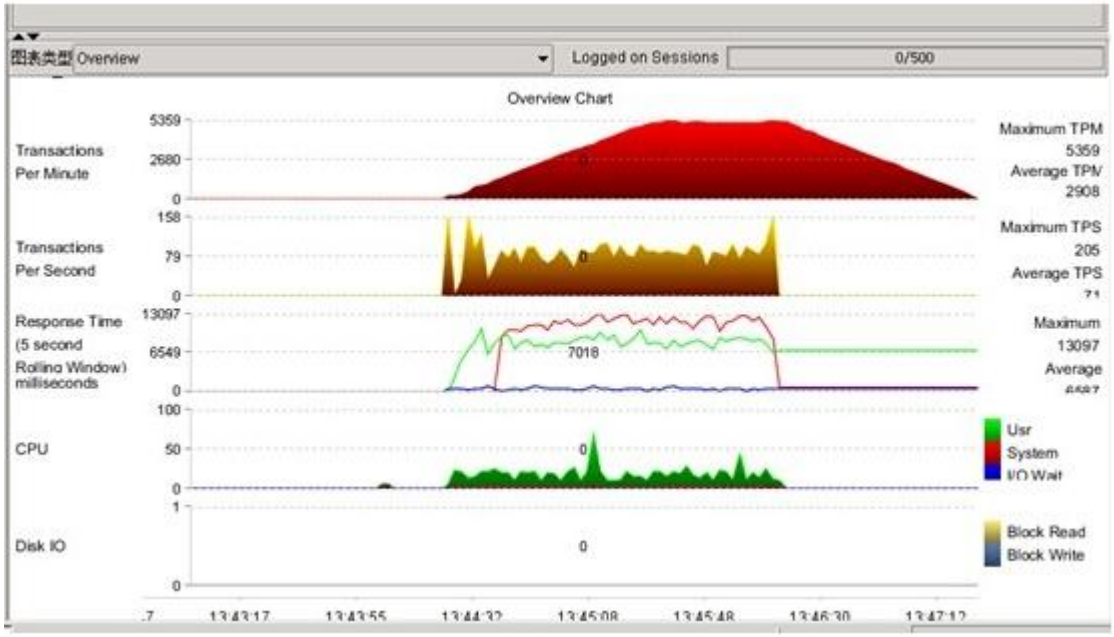
设置 insert，update ，select 的比例



设置完成之后，点击左上角绿色按钮执行测试



测试过程截图



测试结果可保持为 XML 文档，最后查看显示如下

Benchmarks results comparison

Benchmark Settings

File Name	Benchmark Name	Date	Run Time	Number of User	Minimum Think Time	Maximum Think Time	Connect String
results172.xml	"Calling Circle Benchmark"	2012-9-10 16:54:43	0:00:59	400	1	10	//172.16.51.172/SFC3DB

Overview Results

File Name	Transaction Count	Average Transactions/sec	Max Transaction Count	Failed Transaction Count	Nested Transaction Count	Average Nested Transactions/sec	Max Nested Transaction Count
results172.xml	7871	133.41	7750	396	58973	894.61	250412

Response Times

Transaction Name	Response Times (timings in milliseconds)			
	File Name	Minimum	Average	Maximum
New Customer	results172.xml	757	2625	11557
Update Customer Details	results172.xml	450	2043	11478
Retrieve Customer Details	results172.xml	8	135	9533

Database Statistics

Statistic Name	results172.xml

忘记管理员密码的补救办法

作者：舒永春 来源：http://jimshu.blog.51cto.com/3171847/1563207

如果不慎遗忘 SQL Server 的管理员密码（即：遗忘了所有的管理员密码），或者需要强行添加另一个管理员帐号，这时候需要一种补救措施。

SQL Server 提供了单用户模式（也称为维护模式），便于用户更改服务器配置选项或恢复受损的系统数据库。在单用户模式下启动 SQL Server ,可以使本机 Administrators 组的任何成员作为 sysadmin 服务器角色的成员连接到 SQL Server 数据库引擎（实例）。

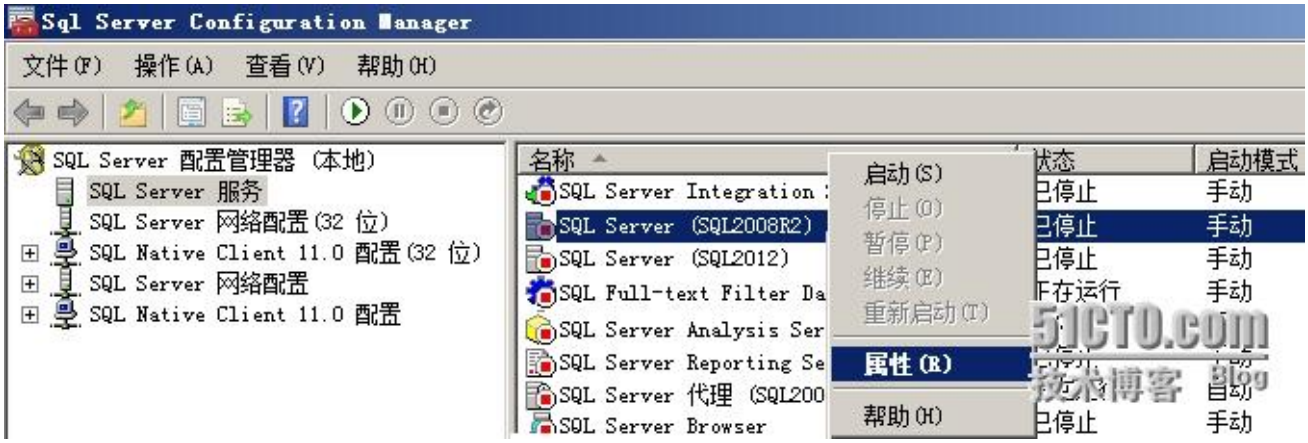
注意：在单用户模式启动之前，请停止 SQL Server Agent 服务，防止 SQL Server Agent 抢占此唯一的连接。

一、启用单用户模式

1. 打开 SSCM（SQL Server 配置管理器）

2. 停止 SQL Server 引擎服务（实例）

3. 修改该引擎服务的属性



4. 添加单用户模式启动参数

（1） 适用于 SQL Server 2012 之前的旧版本

（2） 适用于 SQL Server 2012 及后陆版本

5. 完成上述修改后，启动数据库引擎服务

6. 检查启动日志，确认已进入了单用户模式

```
2014-10-13 13:55:08.95 spid7s      SQL Server started in single-user mode. This an
informational message only. No user action is required.
```

注意：启动日志文件的位置及文件名由“启动参数”的“-e”参数指定。例如：

```
C:\Users\Administrator> notepad "C:\Program Files\Microsoft SQL
Server\MSSQL10_50.SQL2008R2\MSSQL\Log\ERRORLOG"
```

二、越权添加或修改管理员帐户

1. 查看数据库引擎服务的列表（确保下一步操作时不会敲错实例的名称）

```
C:\Users\Administrator> sqlcmd -L
服务器:
    PC2014
    PC2014\SQL2008R2
    PC2014\SQL2012
```

2. 打开命令行窗口，使用 sqlcmd 连接到数据库

```
C:\Users\Administrator> sqlcmd -S PC2014\SQL2008R2
```

3. 执行以下其中一条 T-SQL 语句，添加或修改帐号

（1）添加本地或域帐户到数据库管理员组

```
1> EXEC sp_addsrvrolemember 'PC2014\jim', 'sysadmin';
2> GO
```

（2）添加内置帐户到数据库管理员组

```
1> EXEC sp_addsrvrolemember 'BUILTIN\administrators', 'sysadmin';
2> GO
```

（3）如果遗忘了旧密码，重置密码 （官方不推荐使用 sp_password）

1> Alter Login [BUILTIN\administrator] with password='newpassword';

2> GO

（4）如果还记得旧密码，修改密码

1> Alter Login [sa] with password='newpassword' old_password='oldpassword';

2> GO

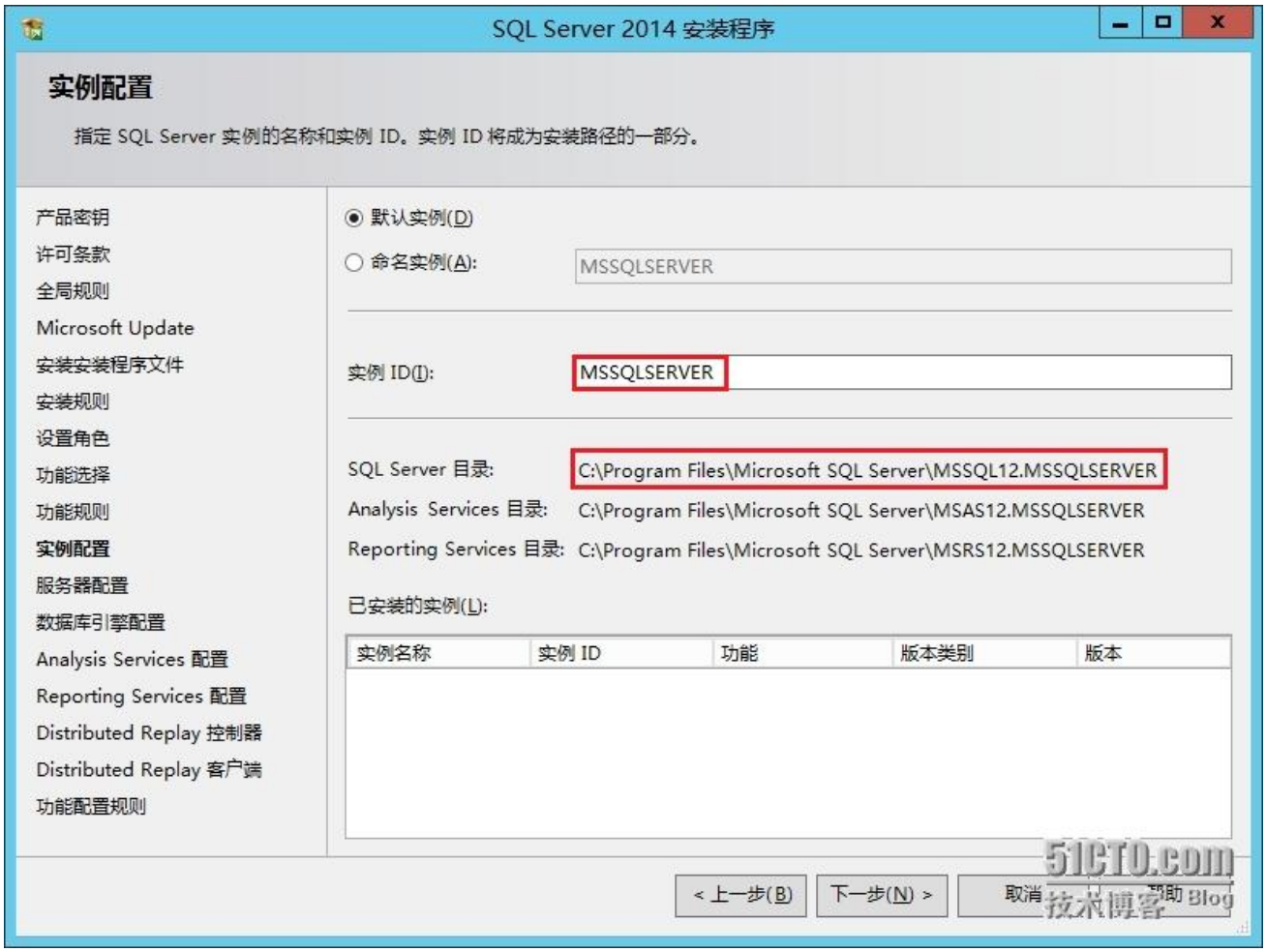
注意：如果 SA 帐户被禁用，则

1> Alter Login [sa] ENABLE;

2> GO

注意:如果服务器身份验证模式仅为“Windows 身份验证模式”,那么需要改为“SQL Server 和 Windows 身份验证模式”，才可以使用 sa 帐户。修改此模式需要修改注册表，注册表项位于：HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\<实例 ID>\MSSQLServer，将 LoginMode 改为 2 即可。

“实例 ID”的命名格式默认为“MSSQL<版本号>.<实例名称>”,例如“MSSQL12.MSSQLSERVER”。这个设置是在安装 SQL Server 时指定的，安装向导将使用这个“实例 ID”创建对应的文件夹。



三、启动数据库引擎服务

1. 打开 SSCM，修改数据库引擎的属性，将“-m ”参数从启动参数中移除。

2. 启动数据库引擎服务，以新建或修改过的数据库管理员帐户连接到数据库。

附注：直接使用命令行启动单用户模式的方法

1. 找到 sqlservr.exe 的路径

2. 复制 sqlservr.exe 到命令行窗口，添加“-m”参数并运行

例如：

```
C:\Users\Administrator> "C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQL2008R2\MSSQL\Binn\sqlservr.exe" -sSQL2008R2 -m
```

启动日志将直接显示在命令行窗口，请检查启动日志，确认已成功启用了单用户模式。

注：还可以强行指定客户端程序。

（1）-m "sqlcmd" 。指定只有 sqlcmd 可以连接到 SQL Server 实例。

（2） -m "Microsoft SQL Server Management Studio -Query" 。指定只有 SSMS 可以连接到 SQL Server 实例。

3. 另外打开一个命令行窗口，即可以使用 sqlcmd 执行操作。

4. 如需停止 SQL Server 实例，只需要在 sqlservr.exe 窗口按 Ctrl-C（甚至直接关闭 sqlservr.exe 的窗口），在遇到提问时回答“Y”。

Do you wish to shutdown SQL Server (Y/N)?

MySQL5.7 加强了 root 用户登录安全性

作者：贺春旸 来源：http://hcymysql.blog.51cto.com/5223301/1569589

MySQL5.7 在安装完后，第一次启动时，会在 root 目录下生产一个随机的密码，文件名为：

```
.mysql_secret
```

登录时需要用随机密码登录，然后通过以下命令修改密码

```
SET PASSWORD = PASSWORD('new password');
```

例：

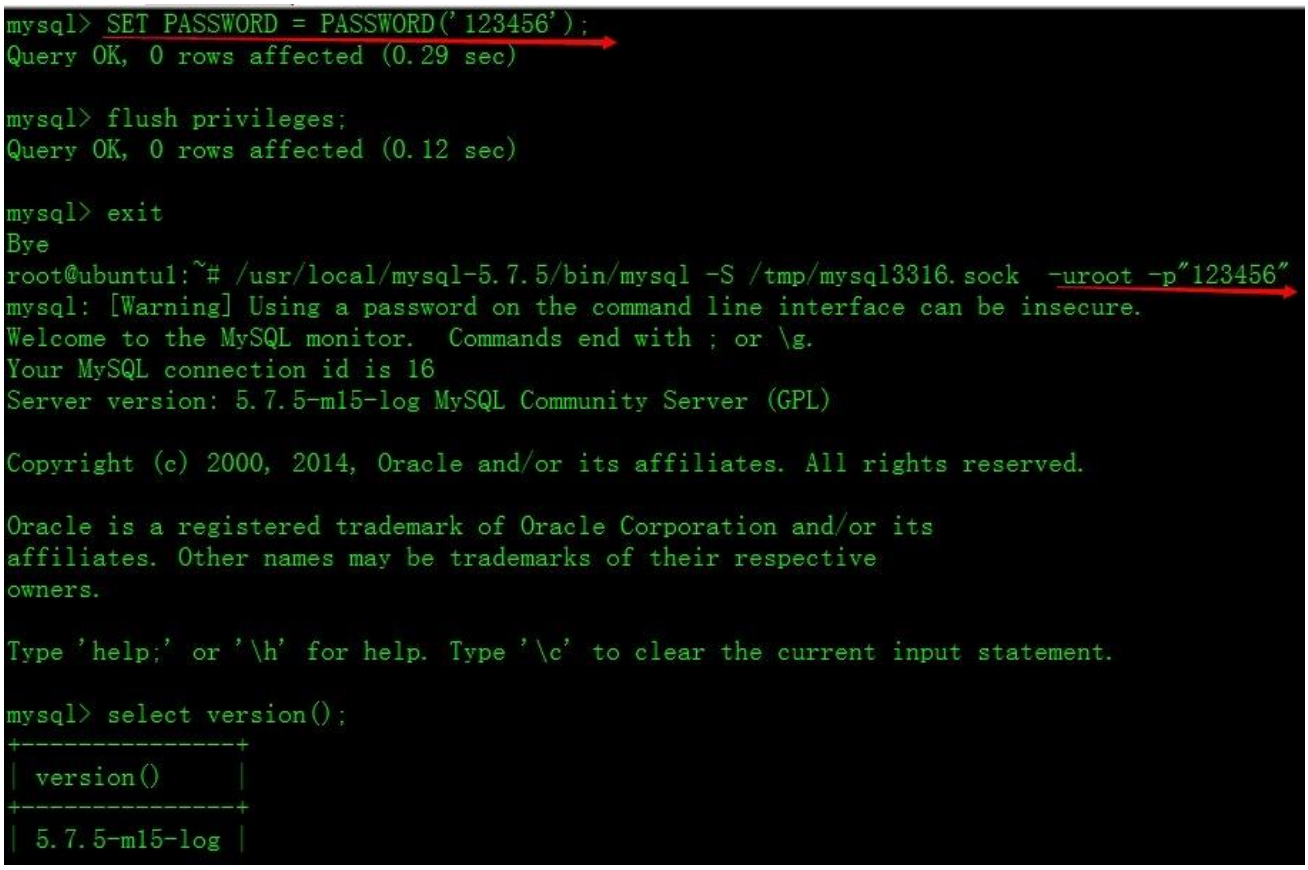
```
root@ubuntu1:~# pwd
/root
root@ubuntu1:~# cat .mysql_secret
# Password set for user 'root@localhost' at 2014-10-26 10:19:21
N>aWmC3E%kkl
root@ubuntu1:~# /usr/local/mysql-5.7.5/bin/mysql -S /tmp/mysql3316.sock  -uroot -p"N>aWmC3E%kkl"
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.7.5-m15-log

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select version();
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement
mysql>
mysql>
```


mysql> SET PASSWORD = PASSWORD('123456');
Query OK, 0 rows affected (0.29 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.12 sec)
mysql> exit
Bye
root@ubuntu1:~# /usr/local/mysql-5.7.5/bin/mysql -S /tmp/mysql3316.sock -uroot -p"123456"
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 5.7.5-m15-log MySQL Community Server (GPL)
Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> select version();
+---------------+
| version() |
+---------------+
| 5.7.5-m15-log |

多年以来，你可找到努力的动力？

作者：杨云 1028 来源：http://yangrong.blog.51cto.com/6945369/1561381

最初的梦想是什么，记忆中有点模糊了，想过成为一名导演、盗墓者、小说作家、流浪者，想开一家饭馆、一家超市，有过很多很多的梦想，但没一件真正去做，当然也没能力去做，甚至都不知道做了有什么意义。经常和朋友争论活着的意义，反正死后终是尘土，人为什么要那么累的努力工作学习？

人活着的意义是什么？有人说为金钱，有人说为关心我和我关心的人，有人说为实现梦想等等。以前我也老是纠结在这个问题上，直到某天在知乎看到一篇讨论人生活着意义的文章，才幡然醒悟。他指出大家讨论这个问题时都把它的次序弄反了，人不是因为某个意义而来到世上的，人活着就是为了寻找他的意义。挨饿受冻了，温饱就是你的意义；内心空虚了，情感就是你的意义；日子紧张了，金钱就是你的意义；迷茫了，找到目标就是你的意义；意义不是一尘不变的，它随着你的需求改变而改变。现在仍迷茫的人就应该努力去寻找自己的意义。

国庆七天，我宅在家里，尽管学习的时间不多，但已不像去年那样，沉迷在游戏里了，因为提不起任何兴趣。以前从没想过有一天我能够离开它，别人也许无法理解网游对我的诱惑力，但沉迷过的人都知道，如今我却做到了，所以觉得自己好伟大。是什么动力让我变成了现在的样子？回首过去，在不同的阶段，有着不同的让我努力的动力。人不是一蹴而就想改变就能改变的，我经过好几年止住下坠趋势，再花好几年慢慢进步，今后的我也会一直进步。

第一阶段努力：如果我所向往的成为现实，我是否能承担起这份幸福？

高中喜欢一个女生，但从来没有向她表白，现在她都已是人妻了，时光过得可真快。曾有过很多次表白的想法，那憋在心里的感觉真的很不好受，但最终都被掐断了，当时想法是：如果成功了，我能负责她一生吗？我不会赚钱，家里也没钱，如果我能给的只是和上辈人一样，让老婆呆在农村，使她无法见识中国不同地域的风土人情、世界上不同的国度，仅自己在外打工赚钱，那表白成功岂不是害了她？(我一直认为恋爱是要结婚的，现在也如是)，如果表白失败了，我为什么还要去表白受伤害呢？

所以我发誓要努力学习，读个大学(尽管不入流)，等功成名就了再向她表白。至于因多年未曾联系，情絮淡了那是后话了，更别说她都结婚了，这都已过去了，至少我没有伤害过她。

第二阶段努力：不想让别人的不幸发生在我的身上。

树欲静而风不止，子欲养而亲不待。这确实是一个很严重的问题，父母都老了，我慢慢的耗到三四十岁，等该有的都有了，那又有什么用？父母都不在了。。。我必须让那一切尽可能提前到来！所以我需要努力，我得突破自己，不能再混日子了。

在培训的时候，有些同学年纪是比较大的，30 多岁，却不得不离开妻儿苦逼的学技术，我内心只有一句话：你年轻的时候干嘛去了？都这么大了还来学这些技术，累不累啊。感触最深的是一个同学老婆在家刚生完孩子，他却不得不离开家人来学习，如果不是经济出现问题，谁愿意花这么多心思学技术？(当然我是很敬佩这些来学习的人的，他们敢于突破现状。至于那些年纪大了，一事无成，还不想下狠心提升自己的，我更看不上了)。每次我想如果自己到了 30 多岁，没什么成就，赚的钱不够养自己的家人，还需要离开家人去学习技术来提升薪资。想想都觉得可怕，我发誓永不让这一切发生在我的身上！

我努力的让自己性格温和、知情达理、易接受新东西、会换位考虑别人的感受。因为我不想以后我的家庭是不和睦的，和朋友的关系是肤浅不至灵魂的。我已见得太多吵架，爸妈的不和睦，哥嫂的不和睦，邻居家的不和睦，无非就是相互不理解沟通少、金钱不宽裕导致的。我趁着年轻努力的改变自己，努力的提升赚钱能力，只为将来这一切尽可能少或不会发生在我身上！

第三阶段努力：想让别人的成就发生在我的身上。

随着见识的提升，见到的牛人也越来越多，为什么他 25 岁有 22k 的薪资？为什么他 28 岁有 28k 的工资？为什么 90 后的他成为了部门经理？为什么 90 后的他成为了公司创始人？不，这一切不是专属于他们的，他们很多也和我一样，来自农村，不高学历，用汗水来构筑梦想。所以只要我足够的努力，这些同样可能发生在我的身上！也许这些离我很远，但正如马云所说，梦想还是要有的，万一实现了呢？所以，学习，从来没有止境，不然拿什么去实现梦想啊。

人天生是懒惰的，这是人之常情，我也经常懒惰；无论梦想多么伟大，在最朴实无华的枯燥与寂寞面前，也变得黯然失色。但没办法，也只有傻逼式的坚持。我始终相信，只要每天进步一点点，多年后，所拥有的东西就会是同龄人的很多倍！

在第一第二阶段的时候，我并不知道我的目标是什么，仅仅是做尽可能的偿试来改变现状；在第三阶段的时候，我才清楚我想要达到一个什么层次。很多人不知道现在做的事有什么意义，也不知道以后要做什么；找不到方向，没有目标，这是每个人都会经历的一个阶段。但是在我们找不到方向时，我们可以换个思路，远离自己不想要的！离你不想要的远一点就代表离你想要的可能近一点；坚持下去，总有一天，你就会清清楚楚的看到自己想要的是什么。

我不想忘记，我经历过的每一天都重要无比

作者：莎吧啦 来源：http://songlisha.blog.51cto.com/6269280/1568385

我不想忘记，我经历过的每一天都重要无比

记录自己生活的点滴，等老了翻看回忆起来，原来那些年，我曾那样的活过。

——题记

2013 年 6 月 9 号，拿到学历证和学位证，我毕业了。

2013 年 6 月 11 号，我和王倩携着大包小包踏上了开往北京城的 T98。由于只提前了一天买票，没有座位，一路站了 5 个小时于下午 3 点到达北京。火车上王倩说了一句：要进京了，去紫禁城。

刚下火车就下雨了，那个时刻我心中对北京也是有些许的惊喜吧。（因为 2010 年暑假来过一次，玩了 7 天，北京留给我的印象就是宽阔洁净的马路，楼很高，挺好的，比郑州好多了） 惊喜只是一点点，更多的是无助感。因为我们俩面对的第一个问题是晚上住哪，当时带了 2000 块钱，还要支撑到在北京找到工作，也不知道自己什么时候能找到工作，也不知道自己的工作会找在北京哪个地方，当时和王倩计划的是先租一个月的房子，等我找到了工作，就在我公司和她公司中间再长期租房。但是房子也不是当天晚上来了北京就能找到的。所以就打电话求助在北京的表姐先住下来。

第二天王倩就去上班，我开始约公司面试。好在王倩有个叔在北京，她去找她叔说了我们的情况，她叔就同意我们先暂时住在他们闲置的一套房子里面，等我找到工作再搬出去（这一住就是 3 个月，我换了 2 份工作，王倩也换了工作）。

我和王倩搬进了她叔叔家，在魏公村的一个老房子里。很清静，市里面人不多，大都住一些老北京。路的尽头有家超市发，还有网吧，除了去面试，就是在网吧投简历（在这个网吧把我的一个红色的金士顿 8G 的 U 盘弄丢了，表示很伤心），然后去超市发买些吃的，都不怎么会做饭，我们俩就整天买速冻水饺回去煮着吃。还会煮各种绿豆汤红豆汤米汤。旁边是北京交通大学，有时候吃过晚饭我们会去学校打印一些简历，顺便散散步。打印非常便宜，1 毛钱一张。

没有面试的时候也会去图书馆看书。王倩下班回来我俩就做饭吃饭看书，打扫卫生，呆了一个月我找到工作后她也辞职了，我们俩就互相提问面试题，她拿着我的 Linux 提问我，我问她什么是面向对象。

北京的 6 月，地表温度能达到 50-60 度。我背着简历，一天面试下来，能从南 6 环跑到北 5 环，还要各种找路，幸亏我在找路这方面比较有感觉。有些公司在比较偏僻的地方，到饭店找不到饭吃，一天吃不上饭，很是郁闷。有时候实在跑一天累了，第二天的面试会全部爽约，好好休息一天。

有时候面试会碰到一些非常让人气愤的面试官，一个技术题不问，随便聊聊就说不满意。最后我问 DH 的 HR 他们为什么要这样，DH 的 HR 告诉我，有些面试官只是想好奇这是个什么妹子来应聘这个工作。（作为一名女运维工程师，我从来没有感受到什么女生受到照顾，相反，性别带给我的障碍是广大男生不会体会到的。）

2013 年 6 月 16 号：今天面试的第一家公司在星光影视园，很摩登漂亮的园区，接待我的 hr 是一位很漂亮的美女，聊了一下没有我感兴趣的职位。第二家面试我的技术主管竟然也是一位美女，这让我真的很意外。技术面试回答的不好，LAMP 的工作流我真心不清楚，FTP 的原理也没有追究过。但是 HR 说想要应届生，问我想要多少薪水，我说 5K。她说下周一给答复。仔细想一下确实是自己在学习过程中缺陷太多，总是懒得去追究事情的原理。改之！

2013 年 6 月 19 号：太累了今天躺了一天，约好的面试也没去。上午一觉睡到 11 点多起床煮了 12 个饺子 3 个粽子，又啃个苹果。然后躺床上看韩剧《灿烂的遗产》看到晚上 8 点（我从来不喜欢韩剧，喜欢此剧是因为觉得女主角像我的秀，真诚，善良，勤劳又会做饭，重点是长的也很像！最后有个非常幸福的结局），接了 8 个面试邀请。明后两天跑 8 家公司，我得给自己找个马达装上！不过还好工作有点眉目啦！接到一个复试邀请！加油加油！

我还记得我第一家面试的公司是完美世界，在大屯，一个我非常喜欢的公司，公司大厦也非常漂亮，我还拍了照片留了念，知道自己也不会被应聘上，但是还是想去试试看，增加一些大公司的面试经历也是好的。

跑了 6 个工作日，面了大概有 15 家左右的公司吧，被 3 家录用了，第一家是刚成立的小公司，公司

自己开发的软件外销，让我负责售后的维护，意思就是要全国各地的出差。第二家是 A 股上市的大公司，公司总部在成都，让我做网络方面的工作，主要就是思科和华为的交换机，还负责公司内网的 3 台 Linux 服务器。第三家是个中等公司，公司总部在郑州，说是有机会回郑州工作，天知道我有多讨厌郑州。

简单了考虑了一下我选择了第二家 ,在建国门上班。第一家小公司也不错 ,但是我觉得自己能力有限，不敢一个人负责。每天从 4 号线魏公村站到西单换乘 1 号线建国门站 ,需要 1 个小时。路上会途径天安门，东单。每次到了天安门都会有很多人下车。我就想着，难道这些人都在人民大会堂上班吗。

从 7 月 8 号到 8 月 20 号在该公司工作了 44 天 ,我没有做过一件事。就是整天傻呆着 ,拿着一本 CCNP 的培训教材看，然后天天望眼欲穿的盼着 Linux 服务器出问题，因为出了问题我才有事情做啊。还每天帮助一个同事喂鱼。

这 44 天过的真是安逸，公司有夜班，我的小伙伴整天值夜班处理网络故障。我跟经理申请我不要上夜班，因为我不会处理网络故障。经理同意了。（我对网络上的事真的是一窍不通，仅限于知道 IP 分段这种层度）。期间还去了山西的五台山旅游了 2 天，各种吃喝玩乐，很是 happy。

直到 8 月 10 号发工资，是一个我非常不满意的数字，1180 块钱，这让我非常愤怒，我自认为我就是整天坐着什么都不干也不能给我这个数字啊。我收到工资短信就直接把我的办公桌收拾干净，准备投简历找新工作。

又面了 3 家，8 月 22 号被 DH 录用了。去 DH 面试是约的某个上午 10 点，到了约定的时间，我不敢给经理请假，我就说不去了。然后 DH 的 HR 开始说我，怎么了？你是不敢请假？ 我最讨厌别人说我不敢做什么，我直接就说那再等一会，下午 2 点我一定过去。然后我就背着包走了，到了地铁 1 号线上想着直接走了影响不好，就给经理打电话说我肚子疼，回去睡觉了。

和大部分互联网公司一样，到了 DH 是先去前台登记来面试的，面试运维工程师，然后前台很懂的拿出来了 9 页试卷让我做题。还说了一句捡会的做。由于会议室里都坐满了面试人员，我就在过道的桌子上满头大汗的做了 2 个小时的 Linux 题。试卷填的密密麻麻。我对面坐着一个来面试 PHP 程序员的女生。她还问我为什么这么多试卷，我说没办法啊，搞 Linux 的是要求编程，系统，网络，数据库都要懂一点的，

每个方面 2 页就这么多了。

做完题运维主管就来给我聊。拿着我的试卷说，填的挺满的啊。然后又问了我 3 个技术题。我都答对了，他又说这个职位要值夜班。每个月上 15 个班，一半白班一半夜班。他问我想要多少薪水，我说低于 4000 不干。他就说你这水平要少了，等会 HR 来问你，你多要点哈。我说那我要 4500 吧。他就说让经理来看看吧。经理下来和我聊了一会，经理问的层度就比较深了，不会简单的问你个填空题，他会给你出个思考题，看你的思路。

然后就是 HR 面试，HR 除了问我有没有男朋友之外，什么问题都问了。你是独生女吗，你在哪住，你爸妈做什么的。用 3 个词形容你自己，你朋友都说怎么评价你。。。等等等。。HR 的最后一个问题是你要多少薪水，我说 4500。HR 说好，2 个工作日之内给你通知。然后第二天就打电话说你不是要 4500 吗。那就 4500 了。明天来上班吧（每个月有 2000 块左右的补助）。

我就直接找经理说不干了 ,经理问我理由 ,我当然没有说工资太少 ,我说跟我理想不符合 ,我想往 Linux 方面发展，这网络太不好学了。经理说，你是找到了新工作吧。我说是的。然后他说再呆半个月吧。

我说不行，新公司下周一就让去上班。然后就放我走了。

经理不太希望我走，大概是刚毕业整天打了鸡血一样的工资激情感动了他把，他还跟公司的行政说我们这特别优秀的新员工可不可以一个月就给转正呢。走的时候挺伤感的，因为一起的几个小伙伴关系都挺好的，对我也不错，经理对我也不错。公司聚会喝酒（新员工必须喝）经理还替我挡酒。但是工资实在是无法接受，根本就活不下去。我对工资的要求一直都是够活下去就行了。

然后就很没节操的下周一直接拎包去了新公司上班。

2013 年 8 月 26 号-2014 年 7 月 3 号。在 DH 干了将近 1 年。这一年 ,除了让我越来越讨厌我的工作，我什么都没有学到（在后来的工作中我发现，其实有学到很多，只是学习是一件潜移默化的事情，很难看出来效果，只有真正用到了，解决问题的时候有思路才会发现，曾经经历过的每一天都不是白过的！）

整天要上夜班，生物钟全乱了，从来不长痘痘的我也长了一脸小痘痘。夜班虽然不做什么事也可以睡觉，但是睡的不安心，毕竟是担着责任的，谁能呼呼大睡呢。然后下了夜班就开始狂睡觉，睡了一天晚上

就睡不着了。整天过的昏昏沉沉，也没有心情学习。更别提什么进步了。。。但是工资确实给的不少了，也不做什么工作。就是熬夜厉害。。有时候困的特别厉害了，手里握着手机，把手机铃声调到最大，有一次响了 57 条报警短信才把我吵醒。凌晨 3-6 点是最困的时候，就在这时段定闹铃。幸运的是这一年我没有出过什么故障。没有给公司造成损失。

直到 2014 年 4 月份。一个朋友说要找工作，找 8000 的。我就想着为什么我不可以。然后就开始投简历约公司面试。直到遇到现在的工作，那天我刚下夜班，本来是和 BIT 约好的面试，但是夜里出了故障，太累了，而且路途也比较远，就不想去面试。HR 非常认真负责的一遍一遍的打电话问什么时候到。又发短信说要路上注意安全啊什么的。我就回去换了比较正式的衣服和鞋子，一夜没睡直接洗把脸就出门了。

熬夜脸肿了一个大包，都快看不见路了。这多影响形象啊。HR 看到我就非常热情，一副就是我的样子。

HR 问我为什么想换工作，我就指指脸上的包说，天天夜班，都成这样了。然后简单聊了几句就问我要多少钱。我说的是 6-8K。当时是想着超过 6K 不熬夜就行。脸上肿的包让我觉得实在不值得这样牺牲自己。她让我跟我现在的部门经理聊，经理说的是让我负责 windows 和 linux 的工作。我说 windows 的工作我干不了，没有这方面经验。当时想着反正这个工作我也不愿意干。就开始狂吐槽。

他问我有什么职业规划，我说我不知道，这一年，我对我的工作已经不热爱了。如果不是让我整天敲命令，而是处理各种部门之间的人际关系，我觉得没有什么意思。我本是开发专业出身，却阴错阳差的来了 Linux 行业。来了之后发现 Linux 这行业挺适合我的。工作的也很开心。解决掉一个问题时是那么的有成就感。但是 DH 的工作却让我觉得不是那么纯粹的技术，更多的是周旋在开发个部门老大之间，Search 部，Seller 部，Buyer 部，风控部。。等等。。还有测试和 DBA。找到他们的错误，然后他们互相推卸责任说不是自己部门的错。。。这让人很无语。。还会被开发部老大骂公司要你们干嘛。

又面试了几家之后，觉得找到自己满意的工作基本无望。就开始学玩 LOL。打算干到过年再跳槽。

5 月 22 号。BIT 的 HR 又给我打电话问我找到了工作吗，我说没有再找工作了。她说他们现在有我想做的工作了，专门招了个人搞 windows，把 Linux 分给我负责，问给我 offer 我愿意吗。我当然愿意。我听到她话里面的意思是非常希望我过去上班。我就说了薪水低于 8K 不行的。然后她让我制作英文简历给外国 leader 看。直到 6 月 3 号接到了录用的 Offer。薪水比我要的还要多，HR 说是因为我太优秀，大家一致决定是这个数字。

6 月 3 号当天我就去找 DH 经理辞职，但是辞职需要一个月才让走。也就是 7 月 3 号才放人。辞职的时候经理说了很多，我印象最深的一句就是，你就算坐上了 Chad（DHCTO）的职位，就算当上了 CTO 又能怎么样呢。女生还是以家庭为主，这行业女生很少，能平衡好家庭的工作的女生更少，你要有自己的生活。

他说我与一年前的那个 Lisa 不一样了，刚来的时候又楞又冲，现在沉默寡言。这也是后来的很多人对我的评价，我感觉我是真的变了，从内到外，所以别人在问我想过什么样的生活的时候，我都觉得这个问题真是极为可笑。

呵呵。生活，不是我想过什么样的。而是我有能力过什么样的。我只想过自己力所能及的生活，我不再做梦。我也曾怀着极大的热情投入到工作中，但是我却发现工作远远不是我想的那样。

北京，怎么说呢，这里机会很多。搞 IT 的还是在北京比较有发展。我也会在等公交车的时候与旁边摆摊的租房卖房中介公司的人聊天。看看北京的房价。5 环，60 平。230 多万。

后记：来到 BIT 这 1 个月，工作的非常开心。领导对我也很重视。也慢慢的找到了自己的方向。工作学习也都回归正常。又开始对人生有无限期待。感谢 BIT，感谢领导的赏识。这一路走来必须是实力+运气。我总觉得自己靠运气多一些。大概是自己的内心一直充满感激的吧。我也会继续揣着这份感激走好今后的路。

当我走在这里的每一条街道

我的心似乎从来都不能平静

除了发动机的轰鸣和电器之音

我似乎听到了他烛骨般的心跳

我在这里欢笑

我在这里哭泣

我在这里活着

也在这死去

我在这里祈祷

我在这里迷惘

我在这里寻找

也在这儿失去

北京 北京

咖啡馆与广场有三个街区

就像霓虹灯到月亮的距离

人们在挣扎中相互告慰和拥抱

寻找着追逐着奄奄一息的碎梦

我们在这欢笑

我们在这哭泣

我们在这活着

也在这死去

我们在这祈祷

我们在这迷惘

我们在这寻找

也在这儿失去

北京 北京

如果有一天我不得不离去

我希望人们把我埋在这里

在这儿我能感觉到我的存在

在这儿有太多让我眷恋的东西

我在这里欢笑

我在这里哭泣

我在这里活着

也在这儿死去

我在这里祈祷

我在这里迷惘

我在这里寻找

也在这儿失去

北京 北京

扫描二维码，加 51CTO 博客为朋友

关注 51CTO 博客微信，打造你的个性！

http://blog.51cto.com

编后语

《51CTO 博客月刊》为 51CTO 博客整理出品

如果您有意投稿，请联系我们
如果您有反馈意见，请告诉我们

联系方式：
Email：blog@51cto.com
QQ: 1173854158

欢迎关注我们：
@51CTO 技术博客